Cloud-Efficient Modelling and Simulation of Magnetic Nano Materials

D o c t o r a l T h e s i s

(D i s s e r t a t i o n)

to be awarded the degree

Doctor of Engineering (Dr.-Ing.)

submitted by

M.Sc. Pavle Ivanović

from Belgrade

approved by the Faculty of

Mathematics/Computer Science and Mechanical Engineering

Clausthal University of Technology

Date of oral examination

20.05.2021

Dean
Prof. Dr. rer. nat. Jörg P. Müller

Chairperson of the Board of Examiners
Prof. Dr.-Ing. Michael Prilla

Supervising tutor
Prof. Dr. Christian Siemers

Reviewer
Prof. Dr. Ramin Yahyapour

Date of submission: 15/02/2021
Date of oral examination: 20/05/2021

Abstract

Scientific simulations are rarely attempted in a cloud due to the substantial performance costs of virtualization. Considerable communication overheads, intolerable latencies, and inefficient hardware emulation are the main reasons why this emerging technology has not been fully exploited. On the other hand, the progress of computing infrastructure nowadays is strongly dependent on perspective storage medium development, where efficient micromagnetic simulations play a vital role in future memory design.

This thesis addresses both these topics by merging micromagnetic simulations with the latest OpenStack cloud implementation while providing a time and cost-effective alternative to expensive computing centers.

However, many challenges have to be addressed before a high-performance cloud platform emerges as a solution for problems in micromagnetic research communities. First, the best solver candidate has to be selected and further improved, particularly in the parallelization and process communication domain. Second, a 3-level cloud communication hierarchy needs to be recognized and each segment adequately addressed. The required steps include breaking the VM-isolation for the host's shared memory activation, cloud network-stack tuning, optimization, and efficient communication hardware integration.

The project work concludes with practical measurements and confirmation of successfully implemented simulation into an open-source cloud environment. It is achieved that the renewed Magpar solver runs for the first time in the OpenStack cloud by using ivshmem for shared memory communication. Also, extensive measurements proved the effectiveness of our solutions, yielding from sixty percent to over ten times better results than those achieved in the standard cloud.

Zusammenfassung

Aufgrund der erheblichen Leistungskosten der Virtualisierung werden wissenschaftliche Simulationen in einer Cloud selten versucht. Beträchtlicher Kommunikationsaufwand, erhebliche Latenzen und ineffiziente Hardwareemulation sind die Hauptgründe, warum diese aufkommende Technologie nicht vollständig genutzt wurde. Andererseits hängt der Fortschritt der Computertechnologie heutzutage stark von der Entwicklung perspektivischer Speichermedien ab, bei denen effiziente mikromagnetische Simulationen eine wichtige Rolle für die zukünftige Speichertechnologie spielen.

Diese Arbeit befasst sich mit diesen beiden Themen, indem mikromagnetische Simulationen mit der neuesten OpenStack Cloud-Implementierung zusammengeführt werden, um eine zeit- und kostengünstige Alternative zu teuren Rechenzentren bereitzustellen.

Viele Herausforderungen müssen jedoch angegangen werden, bevor eine leistungsstarke Cloud-Plattform als Lösung für Probleme in mikromagnetischen Forschungsgemeinschaften entsteht. Zunächst muss der beste Kandidat für die Lösung ausgewählt und weiter verbessert werden, insbesondere im Bereich der Parallelisierung und Prozesskommunikation. Zweitens muss eine 3-stufige Cloud-Kommunikationshierarchie erkannt und jedes Segment angemessen adressiert werden. Die erforderlichen Schritte umfassen das Aufheben der VM-Isolation, um den gemeinsam genutzten Speicher zwischen Cloud-Instanzen zu aktivieren, die Optimierung des Cloud-Netzwerkstapels und die effiziente Integration von Kommunikationshardware.

Die praktische Arbeit endet mit Messungen und der Bestätigung einer erfolgreich implementierten Simulation in einer Open-Source Cloud-Umgebung. Als Ergebnis haben wir erreicht, dass der neu erstellte Magpar-Solver zum ersten Mal in der OpenStack Cloud ausgeführt wird, indem ivshmem für die Shared-Memory Kommunikation verwendet wird. Umfangreiche Messungen haben auch die Wirksamkeit unserer Lösungen bewiesen und von sechzig Prozent bis zu zehnmal besseren Ergebnissen als in der Standard Cloud geführt.

Acknowledgments

The work on this thesis has been an exciting, challenging, yet always inspiring experience. However, it has been made possible only by honorable people that supported me all these years.

I am very grateful to my former supervisor Prof. Harald Richter who gave me the chance to realize this exciting research project and participate in various engaging academic activities. He supported me with his experience, guidance but also encouraged me to pursue independent research excellence. I would also like to express my sincere gratitude to Prof. Ramin Yahyapour and Prof. Dietmar Möller for their invaluable advice and ideas, which brought my work to a higher professional level.

I owe the success of this project to Prof. Christian Siemers, who took over as my primary supervisor and enabled me, with his knowledge, guidance, and generous support, to conclude all my research activities and complete my dissertation. This project is initially granted and financed by a joint venture of SWZ Simulation Center and TU Clausal.

Finally, I wish to thank my lovely wife and my great parents for their continuous support and encouragement. I dedicate this thesis to you.

CONTENT

1 INTRODUCTION

The tremendous technological progress in recent decades of computing power, miniaturization, and production efficiency resulted in the exponential growth of consumer devices and online services. Because of significant manufacturing cost reductions and product availability, the internet became accessible even in most rural areas and allowed for a global interconnection. Consequently, high pressure was put on contemporary storage technologies due to the sheer quantities of newly emerged and, in principle, unstructured data, such as photos, videos, and audio records. In addition, the increasing popularity of social media and the expected massive enrolment of IoT devices in smart homes and cities would only increase the massive system loads. The resulting effect will consequently accelerate the attaining of the upper limits of the present silicon-based storage solutions. The reason for this lies in physical limitations that are almost reached with present levels of transistor miniaturization and increased design complexity. In addition to investment costs, heat dissipation and energy consumption problems pose a significant challenge in maintaining huge server farms due to rising operational costs.

On the other side, emerging technologies such as magnetic nanomaterials provide the ultimate solutions for permanent data storage without the external power supply requirement. With their exceptional storage densities, magnetic nanomaterials are considered as a base for future computer memories and a promising replacement of present silicon-based implementations. However, precise computer modeling of physical conditions occurring on nanoscale requires the engagement of computationally demanding simulations and expensive laboratory equipment, such as scanning tunnel microscopes. Furthermore, time-consuming simulations have to be executed on super or alternatively parallel computers, which are not accessible to broad research communities. Simulators are proven as useful tools because they enable calculating elaborate initial or boundary conditions by simply changing the input configuration. Suppose magnetic nanomaterial modeling could be performed on a more affordable platform. In that case, the results obtained from simulations could significantly

accelerate the progress in this attractive research field and efficiently guide scientists in the most promising directions.

In parallel to the substantial development of computing power, hardware design, and organization on one side and the success of the internet on the other, the new computing model called cloud computing emerged and made a significant change in the data handling, storing, and processing. Cloud computing became a universal platform for on-demand provisioning of computer resources while maintaining the focus on data storage and computing power. It enables flexibility in virtual resource customization and a built-in elasticity regarding the number of active users. Furthermore, cloud computing enables offloading computing and storage resources of companies, research centers, or individuals to the commercial cloud service providers (CSP). Their services range from simple multimedia or data storage to complete virtual computing centers. One of the key benefits of the cloud's pay-as-you-go accounting model is IT cost reduction. For instance, research groups do not need to acquire or maintain their IT-Infrastructure, particularly during dynamic or visionary projects where financing is not strictly defined. Instead, researchers could outsource their IT to the CSP of choice and achieve significant savings in purchasing, maintaining, and updating their hardware and software resources. On the other hand, building a private cloud could make resource sharing very efficient among different institutes, particularly without sacrificing specific user configuration requirements. It also allows unprecedented levels of tuning and modification that could tailor compute resources for each individual project.

Because of these advantages, the idea of using clouds for high-performance computing (HPC) and simulation execution seems very attractive to scientists. Thus, the merging of magnetic nanomaterial simulations and cloud computing appears to be promising as it would allow affordable access to the HPC infrastructure to the broad research groups. Otherwise, research in designing future computer memories would be limited to the science communities with access to expensive and sophisticated laboratory equipment and powerful super or parallel computers. However, both clouds and magnetic nanomaterial simulations have their downsides that have to be resolved before efficient and far-reaching applications in the nanoscale memory research field.

The thesis is organized as follows: In the first chapter, the importance and practical applications of magnetic nanomaterials are explained. It is suggested that a novel simulative approach should use augmented cloud infrastructure for running micromagnetic simulations. Additionally, current cloud communication problems are discussed, together with the benefits of using these distributed systems.

In chapter 2, the state-of-the-art of most promising micromagnetic solvers Magpar [1] and Nmag [2] is given, together with alternatives OOMMF [3] and Vampire [4].

The overview of key technologies and software solutions that are necessary for efficient cloud simulations: hypervisor [5], QEMU/KVM [6], Process Scheduler [7], OpenStack [8] [9], MPI [10] ivshmem [11] [12], as well hardware acceleration technologies, is presented in chapter 3.

Chapter 4 provides extensive micromagnetic solver comparison accomplished by designing the uniform input/output software configuration as the first step. Once the respective outputs are obtained, the two most promising candidates are selected and analyzed in various communication, system size, and scaling scenarios [13].

In chapter 5, Project Methodology, divided into two main sections, is presented. First, the 3-level communication hierarchy is described, followed by corresponding contemporary cloud implementation analyses. In the second section, proposed solutions and communication enhancement suggestions are given for each level of communication: intra-VM, inter-VM intra-server, and inter-server.

In chapter 6, the detailed implementation of proposed solutions is presented in three distinctive stages. In the first stage, the rebuild of inter-VM shared memory - ivshmem is given together with a detailed integration description. Additionally, we have suggested a new, one-sided MPI_Put function in the form of a wrapper code with built-in UIO [14] drivers and spinlock-free synchronization. In the second stage, we have presented complete ivshmem OpenStack integration, as well as cloud tuning methods, such as CPU load balancing, MPI process distribution planning, and NUMA-aware scheduling [15]. Finally, in the third stage, the Magpar rebuild and cloud integration description is given. As in previous cases, we have shown several variations in communication channel usage, thereby demonstrating the importance of ivshmem deployment [16] and inter-server communication enhancement [17].

Chapter 7 presents comprehensive performance test results, described by following implementation stages outlined in chapter 5. We have shown that significant performance improvements could be achieved if a traditional TCP/IP communication channel is replaced with ivshmem. Our results ranged between improvement factors 3-10 [12] and 3-6 for inter-VM (no cloud) and Inter-Instance (OpenStack) communication, respectively [18], and form 1.4 to several orders of magnitude, in the case of Magpar simulations [16] [17]. Finally, the thesis is ended with a conclusion and a comprehensive reference list.

1.1 Motivation

The typical size of the embedded magnetic particle in early recording tapes was approximately 500 nm, while the average grain size of current hard disk mediums is about 8 nm. Because these particles are so small, specific quantum effects such as exchange are starting to affect the material and have to be taken into account during product design. On the other hand, these values are too large to be described and computed exclusively on a quantum mechanical level, which is, in principle, impossible with current simulation models. However, in this intermediate realm between atomic resolution, where quantum effects play a significant role, and the microscopic world with well-described physical laws, the micromagnetic simulations are proven to be very efficient tools with many advantages. First, they allow great flexibility in the variation of initial material parameters and experiment conditions. Second, the magnetization reversal and magnetization distribution processes, which are generally hard to investigate empirically, could be well described with micromagnetic simulations. Finally, to precisely model physical conditions that occur on a nanoscale, the properties of the perspective Magnetic nanomaterials [19] have to be taken into account.

Micromagnetic nanomaterials are the thin layers or small areas ranging from 1-100nm of a) specific alloys, such as samarium-cobalt or neodymium-iron-boron, b) ceramics, such as iron-oxide, or c) elements like iron, cobalt, nickel, and gadolinium.

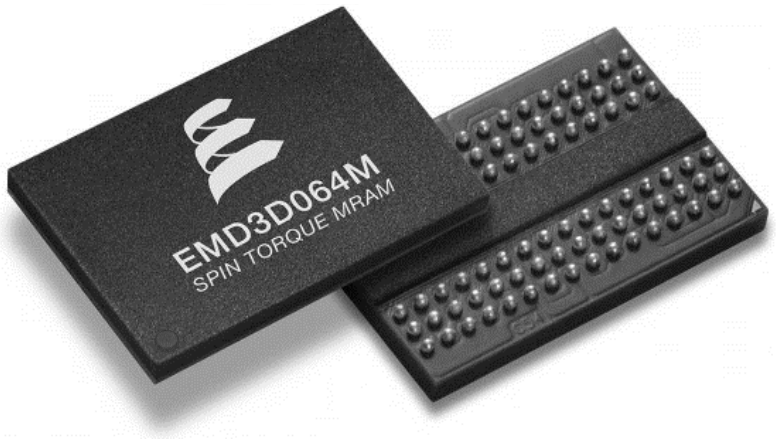

Figure: 1 MRAM chip.

However, the one thing that is common for all these materials is the inclusion of elementary magnets that are even smaller than 1 nm. Depending on the particular alloy or element, these tiny magnets could have different orientations of the local magnetic field, which is usually randomly distributed. However, by applying the external magnetic (write) field, it is possible to simultaneously change the magnetization vector of each elementary particle and achieve the global field orientation. Because of this ability to change the resulting magnetic field of material into the desired direction, magnetic nanomaterials are destined to assume an essential role in advanced technology development. For instance, besides tiny switching units, they could be employed as low power and high-capacity storage devices. This is the main reason why many physicists consider magnetic nanomaterials as inevitable components of future computers, although certain implementations of hard drives, such as magnetic RAM chips (MRAM) [20], already exist on the market (figure 1). One of the key advantages of this technology is enormous savings in power consumption. Contrary to the DRAM chips, which include capacitors that lose their charge over time and require constant “refresh” approximately 16 times per second, MRAM chips do not require a power refresh at all. As a result, power consumption could be reduced up to 99% compared to DRAM chips, which is astonishing.

Another exciting and perspective research in the field of magnetic nanomaterials is the direction of Spintronic or the development of so-called spin-transfer torque (STT) techniques [21]. Spintronic’s main idea is to use the orientation of a single

electron to store a bit value. This technology has the potential not only to create the ultimate storage density but to enable the smallest switching elements in electronics with negligible power consumption. Because of these potential technological breakthroughs, magnetic nanomaterial research is attractive to scientists and has excellent future perspectives.

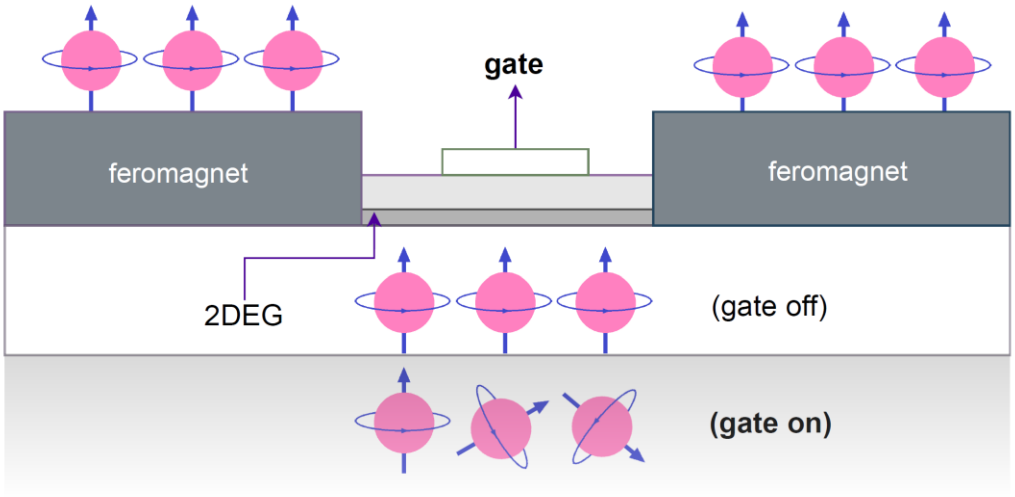

Figure 2: Spintronic - ultimate bit storage in small low-power switching-elements.

Unfortunately, the research in this field requires sophisticated equipment, such as ultra-high vacuum laboratory-setups and scanning tunnel microscopes, in addition to access to high-performance computers. These powerful and expensive computing machines are required for material simulation and magnetic field time/space development on a nano or often pico scale. Because of described financial and organizational obstacles, magnetic nanomaterial research is currently limited to small groups of scientists, which hinders the progress and speed of the commercial technology application.

1.2 Simulative Approach

Although the idea of running material science simulations in a cloud is not entirely new [22] [23], very few efforts have been made in bringing micromagnetic research to this promising computing platform. Also, the solutions proposed in recent years, such as [24], focused primarily on using public cloud infrastructure for accessing GPUs rather than fully engaging available cloud clusters.

However, the GPU approach, in general, exhibits several downsides. First, it requires proprietary hardware as CUDA-based solvers [25] [26] could exploit only particular NVIDIA GPUs, which could be expensive. Second, even when these expenses are mitigated by the sort-term renting of GPU instances, i.g., from Amazon Web Services (AWS), the entire GPU computing capacity is reserved for a single solver run. As a consequence, no other task could be performed in this particular instance. Third, there is a limited number of GPU-capable simulation tools available, particularly in micromagnetics. Fourth, the GPU's effectiveness over CPU is highly ambiguous as the CPU often demonstrates better results, depending on a test case or system size [24]. Moreover, if we consider that this comparison is accomplished between a very modest 4-thread OOMMF setup on hardware with unknown characteristics (e.g., cache configuration) and a modern, massively parallel 1536 CUDA core GPU, it is clear that the CPU approach still has an advantage, at least with current parallelization algorithms.

Conversely, no recorded attempts are made in the private cloud sector, which is unfortunate since a private cloud allows unprecedented levels of access, modification, and optimization for each individual project. Moreover, due to direct host access, a private cloud enables even tuning and enhancing intra- and inter-VM communication, which is essential for efficient simulation execution.

Therefore, to efficiently yet affordably solve emerging challenges in modeling and simulating magnetic materials, a novel approach that employs a private cloud as a computing platform is selected. Consequently, the main direction of this research is not only a replacement of the expensive and often inaccessible parallel or supercomputers with cloud systems but also the creation and distribution of completely established software environments that could be easily scaled, shared, and modified among remote users.

So far, the main reason for the low interest of material scientists in developing efficient parallel codes adapted to the distributed nature of cloud clusters lies in the following contemporary cloud properties: huge communication overheads, variable inter-server latencies, and inefficient system scaling.

First, the processes belonging to the different virtual machines (VMs) could not use the inherited host's shared memory for inter-process communication (IPC) because of VM isolation. This VM design feature is highly inefficient, considering that these processes are visible by the host kernel and have individual access to the main memory. Second, the processes located on the adjacent servers are confronted with several orders of magnitude higher latencies than those on the same server because of lacking direct support for high-speed networks such as Infiniband [27]. Third, the cloud communication stack and virtual network interfaces are not adequately optimized, leading to excessive overheads. Finally, the system scaling becomes inefficient due to the combined effects of issues mentioned above, particularly with a higher number of processing elements.

On the other hand, employing a cloud as a computing tool provides a cost-efficient alternative to the powerful but overpriced hardware setups. For instance, for the small price equivalent to the prepaid mobile subscription, users could rent a Teraflops of computing power using AWS and acquire the hardware only when needed. Moreover, engaging a private cloud provides a similar performance regarding computing efficiency, but more importantly, it enables far better optimization and tuning possibilities as a result of the direct server access. Finally, this research shows that with particular modifications and test scenarios, it becomes possible to elevate cloud computing capabilities to a near-native execution level, which is essential for running complex micromagnetic simulations.

1.3 Project Objectives and Limitations

This project is started with two main objectives: First, it was designed to enable efficient micromagnetic cloud execution, with a particular focus on dynamic time integration and numerical solving of the partial differential equations [28]. The resulting cloud computing platform reflected the creation of a uniform software environment that could be easily shared among remote micromagnetic research

groups and therefore foster sustainable and cost-efficient development in this promising field of material science.

Second, the cloud's inter-process communication should be tuned and enhanced by respecting the 3-level hierarchy identified as intra-VM, inter-VM (intra-server), and inter-server communication. The benefits of this approach are primarily focused on improving micromagnetic cloud simulation efficiency, although the achieved results have a far greater application, as demonstrated in [12] [18] [16] and [17].

However, this project contains certain initial limitations, or more precise, preconditions that should be met in order to be compliant with the stated goals. As a successor of the SimPaaS project [23], this research engages similar guiding principles based on cost savings, operational efficiency, and the open platform approach. Besides utilizing open-source solutions, cost savings also implied excluding the latest chipsets that could undoubtedly lower the absolute execution time values. However, this fact by no means could affect the proposed solutions as they are hardware independent. With these guidelines in mind, the following technologies are carefully selected as a base for further research:

1) OpenStack as cloud OS

2) Message Passing Interface (MPI) for parallelization

3) Open-source micromagnetic solvers supporting MPI for cloud simulation

Since OpenStack [9] is not only a leading open-source cloud OS on the market but also a popular enterprise solution for companies such as IBM, Dell, HP, Cisco, or Red Hat, its engagement in the project was a straightforward decision. One of the key benefits of choosing OpenStack for our private cloud is, of course, the direct access to hardware and software infrastructure, which is not possible with commercial cloud service providers (CSP). Although the closest service model provided by the main CSPs such as Amazon, Google, or Microsoft allows Infrastructure-as-a-Service or IaaS model, the user access and customizations are strictly limited.

On the other hand, MPI is the dominant communication library for parallel execution in the leading world's supercomputers. Moreover, by providing direct access to its source files and having regular upgrades, MPI is indispensable for efficient cloud simulation.

Finally, the project requirements concluded with the open-source micromagnetic solvers capable of MPI parallelization because other solutions, such as [3] or [29], for instance, could not operate in a distributed environment. In addition, open-source packages are free from licensing limitations that could restrict their cloud application. Besides substantial financial savings compared with commercial solutions [30] [31], the benefits of open-source packages are unlimited access to the source codes and the opportunity to inspect the simulation's implementation thoroughly if doubtful results emerge.

2 STATE OF THE ART IN MICROMAGNETICS

2.1 Magnetic Nanomaterials as Bit Memories

In general, the research of magnetic nanomaterials concentrates on the properties and behavior of elementary magnets and their interactions within molecular structures. However, the main focus of the scientists is actually materials that could embrace permanent magnetization when exposed to the external magnetic field. The first group of these materials is called ferromagnetic materials, and their main characteristic is the complete alignment of magnetic moments upon external field exposure. Examples of these materials are metals: iron, cobalt, nickel, gadolinium, and dysprosium, or their alloys, such as aluminum-nickel-cobalt-iron and neodymium-iron-boron. The second group of magnetic materials is called ferrimagnets because some smaller sets of magnetic moments point in the opposite direction, although with an insignificant contribution. Ferrimagnetic materials are, for instance, sintered powders of iron-oxide and barium/strontium carbonate, which further results in the form of a solid ceramic body. In addition, there are other types of magnetism, such as paramagnetism [32], diamagnetism [33], and antiferromagnetism [34], but they have weak responses to external magnetic fields and often require highly sensitive equipment for force field detection.

One of the main reasons for interest in ferromagnetic and ferrimagnetic materials, commonly known as “permanent magnets”, is their persistent storing capacity coupled with very high bit-density. The most prominent example for their application today is a hard drive (HDD), where the rotating discs and read/write heads are coated with micromagnetic materials. Based on the direction of the external write field, elementary magnets distributed on the rotating disc’s surface orientate themselves up or down accordingly, resulting in a bit record. Interestingly, each of these elementary magnets consists of one or more so-called Weiss domains [35], occupying an area of only a few dozen square nanometres. As a consequence, it is possible to achieve TBytes of storage capacity on a disc surface of only several square centimeters. However, this method requires a relatively high current to generate the write-field, which could influence the storage density. High

currents were also the problem of early MRAM write solutions because miniaturization could cause an induced field to overlap adjacent cells, leading to false writing. Fortunately, the new technique called spin-transfer torque (STT) [21], when fully developed, will enable the application of electron spin orientation for storing the bit values. Moreover, there are indications that the spin-transfer torque current consumption could be reduced by a factor of fifty if a perspective composite structure is used [36].

Although there is a clear distinction between the mentioned storage technologies and magnetic materials, there is one physical property that they all comply with. This property is described with a single partial differential equation and could be summarised in the following: every magnetic dipole will sooner or later orientate itself either parallel or antiparallel to the external write field. Therefore, the main factor here is the speed of this orientation process. It designates how fast the memory could be accessed and how many magnetic dipoles are required for reliable bit storage. The answer to both questions in the case of ferromagnetic materials is given by Landau–Lifschitz–Gilbert equation.

2.2 Modeling of Magnetic Materials

The mathematical model applied for magnetic materials was developed over the course of twenty years by three people. The resulting partial differential equation in R^3 is thus called the Landau-Lifschitz-Gilbert equation or LLGE. Equation 1 depicts the first form of LLGE, formulated by [28].

Eq. (1):
$$\frac{\partial}{\partial t}\vec{m} = -|\gamma| \cdot \vec{m} \times \vec{H}_{eff} + \alpha \vec{m} \times \frac{\partial}{\partial t}\vec{m}$$

The vector $\vec{m}$ in equation 1. presents the normalized magnetic dipole moment of the vector field $\vec{M}$, which is created by the elementary magnets in the material. This dipole moment is described by equation 2, where M_s presents the saturation magnetization of the material, which could be considered constant at the observed temperature.

Eq. (2):

$$\vec{m} = \frac{\vec{M}}{M_s}$$

Furthermore, γ is the gyromagnetic ratio of the material, which is constant for the observed material, while $\vec{H}_{eff}$ presents the so-called effective magnetic field caused by the external write field. Finally, α is the so-called Gilbert dumping of the material.

The given LLGE describes the time behavior of the magnetic dipole moment of each elementary magnet influenced by the external write field. This behavior can be illustrated using a mechanical spinner on which an external force or impulse is exerted. As a result, the spinner starts to "gyrate", which means that it rotates around two axes at the same time. This double rotation is also known as "precession". The actual difference between mechanical analogy and the magnetic material is in the external force, which is exerted onto the spinner and must be replaced by $\vec{H}_{eff}$**.**

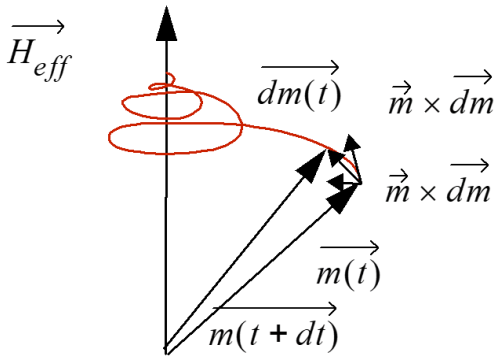

Figure 3: Time behavior of m (elementary magnet) in the external magnetic field.

Furthermore, the spinner must be then replaced by its torque, or in other words, by the dipole moment $\vec{M}$ of the material's elementary magnets. This change results in the gyrating vector field $\vec{M}$ in every elementary magnet. As in the spinner analogy, the gyration of $\vec{M}$ is also dumped. However, only in the magnetic material, the dipole moment $\vec{M}$ of an elementary magnet starts to orient itself either parallel or antiparallel to $\vec{H}_{eff}$**,** which is called "relaxation" of $\vec{M}$. The time development of $\vec{M}$ and also $\vec{m}$ is shown in figure 3.

In figure 4, the same behavior is depicted from the top view.

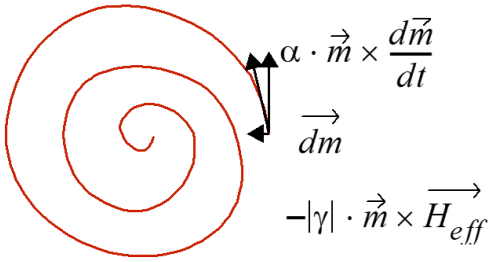

Figure 4: Movement of the normalized magnetic dipole moment m. Top view.

Both figures explain that $\vec{m}$ is moving on a spinal curve into the direction of $\vec{H}_{eff}$. This can be understood by equation 1 in the following way: The first term in equation 1. describes the precession of $\vec{m}$ while the second term denotes how much the amplitude of this precession is damped. To summarize: from equation 1 and figures 3 and 4, it becomes clear that any magnetic dipole in ferromagnetic material orients itself sooner or later in the direction of an external field, thus writing a non-volatile “0“ or “1“ bit into a Weiss domain [35]. The described process is known as magnetization. Because the effect is permanent magnetization, the written bit can be read-out later by a nanoscale read-head. Finally, with the given explanations, the mathematical model that was used is briefly outlined, and the reaming chapters will be focused on simulation software.

2.3 Micromagnetic Solvers

During the literature study, only several open-source micromagnetic software packages capable of solving LLGE were found [1] [2] [3] [4] [25] [29] [37]. Moreover, further filtering of available solutions with respect to the parallelization and possibility of distributed cloud execution reduced the number to only a few solvers. The majority of the remaining packages were designed for sequential execution, which is not suitable even for single multi-core computers. Also, there is a solution based on OpenMP [38], but this application programming interface (API) created for shared memory multiprocessing cannot be used in the cloud because shared memory does not exist between servers. Finally, only a fraction of the remaining solvers were developed using a message passing interface (MPI) [10], which is a standardized interface for large-scale parallel applications and a prevailing communication library inside parallel and supercomputers. In principle, MPI branches into two main implementations, OpenMPI [39] and MPICH [40], but many variations based on the original code, such as MVAPICH2 [41], for example, exist as well. In order to lunch parallel applications, MPI requires the utilization of the so-called process management system, which further engages multiple daemons, such as ssh, rsh, and slurm. However, the majority of MPI-based micromagnetic solvers operate with an obsolete process manager called the multi-purpose daemon (MPD) [42], which has low performance and poor documentation.

Finally, only three open-source MPI-based micromagnetic solvers were found, which are Magpar [1], Nmag [2], and Vampire [4]. Each of them is developed on top of many publicly available numerical and data processing libraries and therefore combined into complex multi-purpose software packages.

In the following chapter, a brief overview of Magpar, Nmag, and Vampire is given, complemented with the evaluation of solver efficiency in the cloud environment. Additionally, to calibrate and set up the appropriate test environment, a fourth open-source micromagnetic solver called OOMMF [3] is included, but its capabilities are limited to a single server shared-memory execution only.

2.3.1 Nmag

Nmag is an open-source finite-element micromagnetic simulation package developed at the University of Southampton in the UK. It is implemented in C and uses Python as a base for the user interface. Nmag engages mpich2 to achieve parallelization, while the mesh input geometry is generated by external tools such as Gmsh [43] or Netgen [44]. Also, it applies the finite-element (FE) method for magnetic material discretization, which is achieved by dividing the simulated system into small tetrahedral shapes that are presumed to have a uniform magnetization vector. Also, the size of each FE is not fixed, and it could present a curved object with great accuracy. On the other hand, the finite-differences method (FD) divides the simulated object into small cubes of equal lengths, but it could achieve the discretization accuracy only in square systems. Furthermore, Nmag uses various tools for post-processing and can also efficiently store the output data in gzip files.

However, similar to other micromagnetic solvers, the development of Nmag is aborted a decade ago, and with recent Linux updates, one cannot guarantee its operability. It also applies to the obsolete mpich2 libraries, which lack the optimization of newer process managers, such as Hydra [42] or the latest communication channel called Nemesis.

Finally, Nmag was successfully ported to the cloud, but it could not match its leading contender, Magpar, particularly when it comes to execution efficiency for the same set of input parameters. On the other side, Nmag installation is much more comfortable, and solver utilization requires less technical knowledge because of its optimized and well-organized input script.

2.3.2 Magpar

Magpar is an open-source micromagnetic software package developed at the Vienna University of Technology. It is written in C++ and uses externally generated mesh geometry files as input. Magpar is designed for parallel solving various micromagnetic problems such as uniaxial and cubic anisotropy, exchange energy, magneto-static interactions, and external field calculations. It also supports both

static energy minimization as well as a dynamic time integration, which is necessary for solving LLGE. This was achieved by engaging many highly efficient numerical solutions for stiff and non-stiff problems, which is very important for HPC cloud simulations.

In addition, Magpar consists of many libraries for geometry modeling, as well as data post-processing. Because of its flexibility, fine-tuning, and mesh configuration, it could achieve high levels of discretization accuracy, particularly in modeling spherical or curved systems. Magpar is also a highly portable solver as MPI libraries allow not only single PC execution but also parallel or even supercomputer enrolment. On the other side, its main competitor Nmag could not provide matching efficiency nor scalability, especially in a parallel cloud environment.

However, Magpar has its downsides. First, parallelization is based on the obsolete mpich2 standard that requires the MPD process manager, which is an old and non-optimized predecessor of the current system called Hydra [42]. Second, it demands the external finite-element mesh generation that has limitations in the discretization of cubic shapes. Third, because it is built on many old or external libraries that no longer exist in their former repositories, it is not easy to install Magpar. Finally, the Magpar rebuild is getting more challenging with every Linux update because software maintenance was stopped almost a decade ago.

2.3.3 Vampire

The Vampire is an open-source project for the simulation of magnetic nanomaterials on the atomic scale. The goal of this project was to bridge the gap between micromagnetic approximation and the actual molecular structure of the materials. Therefore, a set of new physical effects such as spin transport or exchange bias could be simulated with great precision but with a cost of additional system complexity and computation overheads. On the other hand, Vampire engages the latest MPI libraries for efficient parallelization and does not require external tools for input-geometry and mesh generation. Also, setting up a simulation is made very easy by merely editing key-value parameters in the single input textual file.

Similar to other solvers, Vampire could also simulate dynamic time integration of the LLG equation. However, it requires several orders of magnitude smaller time steps to achieve accurate results. Moreover, if a time step scale is not set on a picosecond level, the output results are so distorted that the whole measurement has to be discarded. Therefore, the complexity added by modeling crystal structures on a pico time scale sets the Vampire application beyond our project scope. Finally, to achieve output precision comparable with other micromagnetic solvers, Vampire requires at least three orders of magnitude higher cloud execution times, which is unacceptable.

2.3.4 OOMMF

OOMMF is an open-source micromagnetic research project of the US National Institute of Standards and Technology. It is developed in C++ with the goal of creating a scalable, object-orientated tool for efficient modeling and simulation of micromagnetic systems. OOMMF uses Tcl/Tk scripts as a user interface, making it portable to the most common operating systems today, such as Windows, Linux, or MAC OS. The software package of OOMMF is not compiled as a single program, but it instead uses a series of specialized codes for each specific task of the micromagnetic system. Therefore, this segmentation enables a modular approach concerning further developments and code improvements, making it attractive to many research communities.

Another aspect of OOMMF is the implementation of the finite-differences (FD) method, which does not require external mesh generation. However, although the FD method simplifies the input set of parameters, many curved or spherical systems could not be accurately modeled without increasing the system complexity. Finally, OOMMF uses OpenMP API for parallelization by establishing internal TCP/IP communication. Unfortunately, the current implementation does not support MPI, which is required for running simulations in a cloud. Therefore, OOMMF could be useful for comparing single host execution and calibration of input/output parameters of other solvers, but it is not suited for cloud computing.

3 BACKGROUND AND RELATED TECHNOLOGY

3.1 HPC in a Cloud

Typically, using standard cloud distributions for high-performance computing (HPC) produces significant performance downgrades, particularly in computationally demanding simulations. The main reason for this lies in a low data-exchange efficiency, which is determined by the bandwidth and latency between communicating parts of a parallel code. Depending on cloud configuration and test setup, several factors have a crucial influence on the computation performance. First, in the case of a single VM, the factor that predominantly determines the bandwidth and latency outputs is overhead, generated during communication between logical processing units called virtual CPUs or vCPUs. The second scenario refers to the inter-VM communication that occurs on the same host server. In this case, besides overhead, latency plays a significant role as VM isolation prevents engaging the host's shared memory. However, this could be changed by introducing ivshmem, which will be discussed in detail in the following chapters. Finally, in the case of inter-VM communication, where VMs are located on adjacent or remote servers, latency is the main factor that determines IPC efficiency.

Unfortunately, even with the latest hardware interconnect solutions such as Infiniband, inter-server communication remains a huge bottleneck simply because cache and RAM access speeds are several orders of magnitude smaller than the latency of any wired network medium. Considering that most servers are still coupled with slow Ethernet connections, these performance differences are even more exaggerated. In addition to the bandwidth and latency, the scaling of many simulation codes is not very good because of slow speedup, even if the number of computing elements is significantly increased. As a consequence, running the HPC applications is not recommended on a standard cloud and, in some cases, not even possible. These are the main reasons why cloud was not considered a viable alternative to parallel or supercomputers in the past. Fortunately, open-source cloud solutions such as OpenStack leave much room for improvements, particularly in addressing server overhead and communication latency issues. It

will be shown that with certain cloud modifications, tuning, software enhancement, and targeted hardware updates, an efficient computing platform could be created at low costs.

3.2 OpenStack – Cloud Operating System

OpenStack is an open-source cloud operating system designed for managing and controlling compute, network, and storage resources via a web-based dashboard or command line API. It provides a comprehensive set of services, features, and tools, which are regularly developed and updated every six months. Although different open-access cloud platforms, such as CloudStack [45], Eucalyptus [46], or OpenNebula [47], appeared in the recent decade, OpenStack is still the largest open-source cloud OS in operation today.

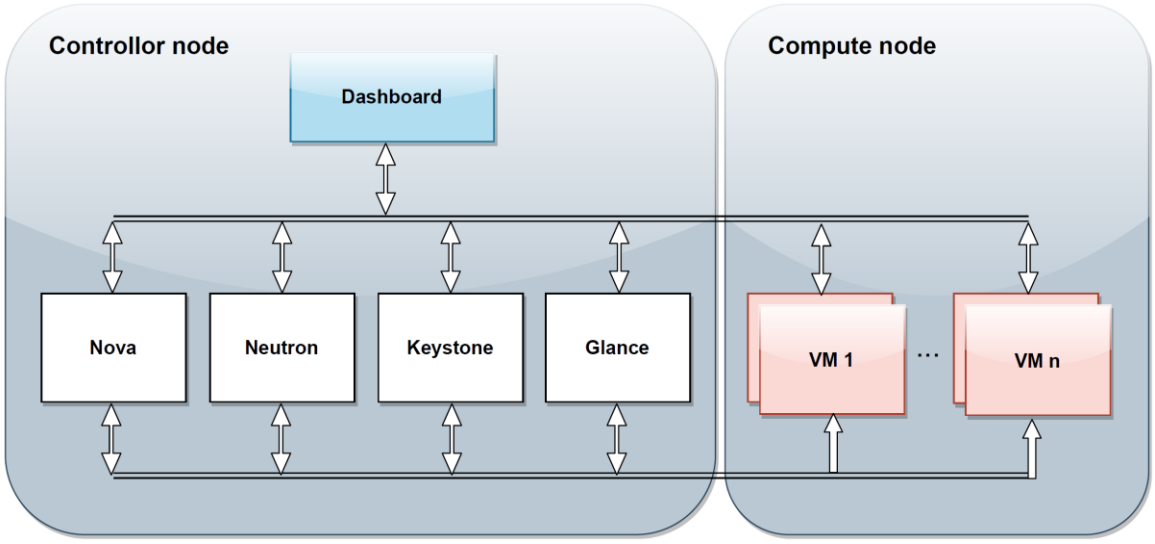

Figure 5: Basic OpenStack services.

Typically, OpenStack cloud consists of at least two distinct hardware units: Controller node and Compute node, and a minimum of five software components (figure 5). The Controller node hosts essential services, such as Nova [48] and Horizon dashboard [49], while Neutron [50] plays a prominent role between different Compute nodes. The primary function of Nova service is VM scheduling, whereas Neutron provides for the virtual network between VMs, physical servers,

and the internet. Storage service Glance [51], for instance, runs on both Compute and Controller nodes and enables virtual storage capabilities. Finally, the security of virtual machine access is achieved by Keystone [52], which was carried through the authorization and authentication of different services.

Finally, OpenStack services mentioned above provide a bare minimum for establishing the private cloud project, at least from the software point of view. On the other hand, all services and software components, including Controller and Compute nodes, could be installed on a single physical machine. In that case, the basic idea of the cloud and distributed computing is losing its original meaning and purpose, although the user would not see the difference. Therefore, to keep the formal cloud definition from a hardware perspective, it is needed that the OpenStack project contains at least two servers, one for Controller and the other for a Compute node.

3.2.1 Horizon

OpenStack Horizon is a web-based graphical user interface for accessing, deploying, configuring, and managing cloud services and computer resources. It provides a convenient way for inexperienced users to manage their cloud projects without requiring profound technical knowledge. Based on the access privileges, the Horizon dashboard differentiates two main modes, which are divided into user and admin accounts. Also, it provides developer tools for VM creation and access automatization by using native OpenStack API. In addition to in-house applications, Horizon allows the customization of current and the introduction of third-party services by editing existing Python classes. In this way, companies and advanced users could completely customize their OpenStack experience. On the other hand, the Horizon menu typically provides a smaller subset of possible settings available in the Nova command-line interface (CLI), and in case of specific or custom configurations, manual input is often required.

3.2.2 Nova

Nova or OpenStack Compute is a cloud computing resource controller and the main component of Infrastructure-as-a-Service deployment. It statically allocates newly created VMs to random servers by applying a predefined set of configurable filters. In other words, Nova performs the so-called meta scheduling, which occurs on a cluster level and not on a host OS plain, where vCPUs are allocated to the physical processing units (CPU, cores). As mentioned earlier, OpenStack Nova provides the well-documented CLI and enables full control of Compute service parameters and settings. However, managing OpenStack exclusively through the CLI requires higher technical knowledge, making it suitable only for advanced users.

In Linux, Nova engages QEMU/KVM hypervisor for VM deployment, but other VMMs [5] such as VMware and Xen are supported as well. To implement the VMs or instances, as they are referred to in OpenStack terminology, Nova uses libvirt API [53] that is built on top of QEMU/KVM. Libvirt is an open-source project that provides support for creating and managing multiple virtualization platforms. In addition to virtual machines, Nova supports Linux container technology (e.g., LXC) [54] as well as bare-metal servers. In recent years, as a growing number of companies are switching their business models into clouds, service performance monitoring becomes increasingly important, particularly in tracking Nova parameters.

3.2.3 Neutron

Neutron is the OpenStack manager responsible for the creation, configuration, and deployment of all virtual network components such as IP addresses, virtual switches, routers, vNICs, and VLANs. It allows for creating both static (DHCP) and Floating IP addresses, which are often used for external VM access. In addition to access from external networks, Floating IPs allow for dynamical rerouting to different virtual resources if VM maintenance or rebuild is needed.

In general, VM-interconnect in OpenStack occurs via the IP layer (ISO layer 3) or VLAN (layer 2), depending on user preferences and system configuration. In

addition to the local VLAN connection between project VMs, a standard OpenStack setup also includes an internet connection, which is essential for guest OS updates and software installations. Furthermore, in order to allow massive system scaling and multi-tenancy project organization, an option to engage software-defined networking (SDN) [55] is given to the cloud administrators. SDN is useful because it tries to centralize the traditionally decentralized network architecture and make network management easier by dividing the network components into data and control planes.

Besides in-house development, OpenStack often supports the integration of third-party solutions such as Open vSwitch (OVS) [56], which is currently an integral part of all Neutron distributions. The OVS role in OpenStack network deployment is vital because the critical network selecting decisions occur in its software-based switches, known as Bridges in Linux. For instance, the OVS Integration Bridge (br-int) is responsible for deciding if the data traffic takes place on the same physical host or the traffic flow should be directed to another server. However, the introduction of different interfaces and virtual switches produces an additional overhead, which is not suitable for HPC. In later chapters, the connection between OVS and the generated overhead will be given in greater detail. Last but not least, Neutron supports creating and managing special services such as virtual private networks (VPN), firewalls, and load balancing. Their role is to improve cloud security and to redistribute the system load better.

3.2.4 Glance

In order to give a better perspective of Glance's role and the features in OpenStack, two similar projects will be briefly described. The first one is called Cinder [57], and it presents a distributed file system for disk records by creating virtual hard drives. In OpenStack, virtual drives are referred to as "Volume", while the technology behind data records is known as block storage. As the name suggests, Cinder's storage units are blocks of data with a fixed length of 2, 4, or 8 KB, depending on the respective disc formatting. The second service is called "Swift", but instead of using block storage, it is designed to operate with objects as storing units. The main difference between the two recording technologies is that objects, besides user data, contain the additional information known as metadata.

The metadata's primary purpose is the provision of extra details to the user data, which can be further used for better organization and data structuring. Moreover, the object storage allows backing up of the whole Swift volume with a command called "snapshot". Snapshot is a very useful feature of OpenStack as it creates the exact copy of a virtual disc and stores it in a single, separate file.

With this brief overview of the two leading storage technologies, the Glance service features could be given. Similar to Swift, Glance uses objects as storing units but records the data in a different format. Glance engages the so-called JSON format, which is organized as key-value pair entries and acts as a repository for the "Image". In OpenStack, the term "Image" is used to describe the single file that contains the record of a whole virtual machine. Based on requirements, the user could simply launch a new VM by selecting the appropriate Image from the repository, which could be either a downloaded version of the selected OS or a stored snapshot. Finally, the described utilization of snapshots is significant for achieving high system scalability and effective parallelization as it allows quick and unlimited multiplication of the existing VMs. Moreover, it provides not only considerable savings in time but also enables efficient system backup and resource management.

3.2.5 Keystone

Keystone is a vital identity management service that provides API for authorization and authentication of OpenStack clients and services. Although the standard method for user authentication occurs by creating and storing RSA key pairs, OpenStack left the option for using username/password credentials. On the other hand, the authorization of OpenStack services is performed by activating the short-term tokens, which is the standard method. However, recent software architecture changes allowed for future applications of other external mechanisms, such as openID [58] or oAuth [59]. Finally, Keystone's end-user interaction could be as simple as copying the RSA public key into the designated field. For instance, suppose the Horizon dashboard is used for cloud access. Because Keystone user setup is typically needed only once per project, it provides a simple but efficient client/service identity management method.

3.3 Virtualization

In computing generally, virtualization refers to the creation of a virtual version of actual physical hardware. This software emulation provides not only virtual compute components such as CPU or memory but also emulates disks, devices, and networking elements. When observed from a cloud perspective, virtualization brings multiple advantages to the end-user. First, it creates no difference for the users compared to standard hardware access and usage because they are not aware of the virtualization. Second, a cloud allows improved system stability since the malfunction of one VM does not influence the operations of others. Third, by enabling the "snapshot" of the file system into a single file, the cloud provides easy system backup. The same principle applies to system scaling because these snapshots turn complex installations and system build-ups into a few mouse clicks. Finally, cloud infrastructure enables easy migration, whether it is needed for maintenance or server consolidation [60]. Besides these general advantages, cloud virtualization has other benefits, but they depend on the virtualization type.

In principle, one can differentiate between two main types of cloud virtualization: virtualization using a cloud service provider (CSP) or virtualization via a private cloud. Both approaches have their advantages and advantages, which will be shortly given. The CSP enables lower costs while simplifying the management and maintenance of the cloud for the end-user. The low costs come from renting computing resources from CSP and its central role in overall system management. On the other hand, the private cloud gives much flexibility in system configuration and modification, and therefore it is more suitable for research and development projects. This notion mainly refers to HPC research, where significant system changes are expected and welcomed.

However, virtualization comes with significant disadvantages, which have to be addressed in order to turn the cloud into an efficient HPC platform. As mentioned earlier, the number one issue in cloud computing is the vast software overhead that emerges from hardware virtualization and cloud network complexity. Therefore, inter-VM communication suffers a great deal of inherited inefficacy and a lack of proper optimization, particularly in the domain of shared-memory IPC. Ultimately, unsolved communication problems lead to the requirement of

expensive chipsets and hardware accelerators, such as SR-IOV [61], for instance, which are often inaccessible and further increase the system complexity.

3.3.1 Hardware Virtualization

Hardware virtualization is the virtualization mechanism that allows the simultaneous execution of various operating systems on a single hardware unit. By providing users the abstract computing platform, hardware virtualization enabled multiple OSs to run independently and unmodified on the same physical machine. In principle, there are three main subtasks of hardware virtualization: CPU, memory, and IO virtualization. However, all three tasks must be accomplished simultaneously in order to provide a virtual server or virtual PC, which is running on the underlying physical hardware.

3.3.1.1 CPU Virtualization

CPU virtualization is a software emulation process that allows one physical CPU to act as multiple virtual CPUs in a server or PC. However, this approach poses a significant challenge because software emulation realized by QEMU exist in User-Space only, while particular core instructions demand system privileges. Those cases require the intervention of system-space modules, such as KVM, in order to solve access privileges violations. Additionally, CPU virtualization requires the creation of virtual translation look-aside buffers (TLB) [62] inside the virtual memory management unit (MMU) [63]. The main issue is that virtual TLB requires frequent updates, and therefore it is mandatory to have hardware accelerators (IOMMU) to achieve effective virtualization performance.

Finally, by applying the concept of full virtualization, it is possible to emulate different hardware vendors but at the cost of performance degradation.

3.3.1.2 Memory Virtualization

Memory virtualization is a process of creating a virtual main memory for virtual machines and their guest OSs. Although this description seems straightforward,

the fact is that the main memory is already virtualized by the host OS, which means that the VM implementation implies the process of double virtualization. The double virtualization is performed as follows: First, the virtual guest OS addresses are translated into virtual host OS addresses. Then, the host OS addresses are translated into the physical addresses of the main memory. In other words, the described process of double mapping occurs between the physical guest OS addresses and physical host OS addresses, which are linked to the real hardware.

However, double virtualization has performance costs. It requires that each guest OS access to the physical memory or physical device address spaces has to be intercepted by the hypervisor (KVM) and further processed by the emulator (QEMU). The same principle applies to all guest OS interruptions and page fault exceptions. This is necessary because the values of the VM page-tables, interrupt-vectors, and memory registers do not reflect the real (physical) conditions and could differ from one VM to another. Consequently, the hypervisor's intermediate interceptions are slowing down the system performance and making the task handling very complicated.

3.3.1.3 IO Virtualization

IO virtualization is the process of mapping the virtual IO resources onto the real peripheral devices. However, this process often requires time multiplexing of the physical devices that are not designed for this purpose. In hosts with several VMs, for instance, multiple virtual devices are competing for one real device that supports only serial access. Consequently, the corresponding multiplexing has to be performed, which is the hypervisor's role. Therefore, IO virtualization aims to create an illusion of uninterrupted guest access to a single physical device, which requires additional IO queue management and novel scheduling mechanisms. These additional mechanisms needed for VM memory management add a high complexity to the virtualization in general, which is critical for performance. As in the case of CPU and Memory virtualization, the usage of hardware-assisted virtualization is thus mandatory because the pure software solutions are incredibly inefficient.

3.3.2 Paravirtualization

Paravirtualization provides for a highly efficient software solution for the reduction of virtualization overhead. It relies on replacing standard guest OS device drivers with the new operating system calls, which are very similar to the host OS counterparts. Because of the design similarities, paravirtualization enables the indirect execution of these system calls, although with the cost of modifying the guest OS source code. Sometimes, these altered guest OS device drivers are called stubs because their primary function is to forward guest call parameters to QEMU and retrieve QEMU/KVM hypervisor results.

Another reason for using modified system calls is potential interference during guest call execution, particularly in a multi-VM scenario. Therefore, to enable smooth inter-system transition, stub device drivers need to have the same API as the original Linux drivers. For instance, KVM provides alternative device drivers using “virtio” [64], which is a Linux kernel module with corresponding system privileges. However, stub calls do not execute anything because they operate within guest OS, which means they do not have access to the real hardware.

Finally, the main benefit of the paravirtualization is the reduction of interrupt interceptions because new guest OS drivers do not execute any privileged instructions. Instead, they are forwarded to the emulator (QEMU) and executed directly in the host OS, saving many CPU cycles. As a result, the software stack between VMs becomes more simplified, which improves communication efficiency and boosts virtualization performance. These are the reasons why paravirtualization is indispensable for cloud tuning and optimization.

3.3.2.1 Virtio-net

The Linux hypervisor contains two primary interfaces: one used by the corresponding emulator (QEMU), and the other, called virtio, which provides an API for paravirtualization. This approach results in a more efficient IO virtualization than full software emulation because it eliminates unnecessary system call interceptions. Besides serving as API, virtio is also a library of paravirtualized device drivers in guest OS. For instance, virtio-net is the paravirtualized device

driver for a virtual Ethernet card (vNIC). In terms of performance, inter-VM communication can profit from virtio-net if a Berkeley socket, for instance, sends the call results directly to the calling virtio-net. Thereby, the driver is aware that it is executed in a VM, and it actively cooperates with its emulator. With virtio-net, the hypervisor does not intercept guest device-driver access to the emulated PC devices because they are not performed. As a result, virtio call parameters are directly forwarded to the QEMU.

Similarly, the Berkeley send() call is not mapped onto a vNIC in guest OS either. Instead, it is placed by QEMU into the virtio-net send queue (“virtqueue”) in the host OS. Virtqueues are in principle less complex and thus faster than vNICs because they are only buffers.

It is also worth noting that the paravirtualized guest device-driver is called the front-end driver, while the modified host Ethernet driver is referred to as the back-end. In addition, the original host OS back-end drivers cannot be used in virtio-net because its input is a Linux-specific data-structure called sk_buf, while the front-end driver outputs virtqueue entries. Nevertheless, the virtio library contains both the front-end and the modified back-end drivers.

The first disadvantage of virtio is the necessity of TCP/IP engagement. The second downside is a requirement that virtqueue entries must be handled by two QEMU processes, one for the source VM and the other for the target. Also, for each data frame sent by the physical Ethernet card, QEMU has to make a KVM system call. As expected, this could be a time-consuming procedure because it requires a full process context switch and the reload of all MMU page table entries.

3.3.3 Hypervisor

A hypervisor presents a layer of software located between virtual machines and host OS that enables visualization of all computer hardware components. The main tasks of each hypervisor are the creation, activation, managing, and stopping of VMs, which are processes commonly known as hardware emulation. In order to successfully and efficiently accomplish the given tasks, each hypervisor requires the operating system privileges. For instance, the hypervisor should execute all CPU commands and have root access to all memory regions, devices, or system

files. However, because one of the main perquisites of hardware emulation is visualization performance, most VM instructions are executed directly by the host CPU. Without this feature, the efficiency of VMs would be so low that modern cloud technologies could not be established. The second important task of the hypervisor is the hardware multiplexing or sharing the host's computing resources between multiple VMs. This means that each guest OS runs unchanged and operates as it is exclusively executed on the underlying hardware. In other words, the emulated or virtual hardware serves as an interface through which the guest OSs could access the same physical CPUs, memory, or devices, although they are entirely unaware of each other.

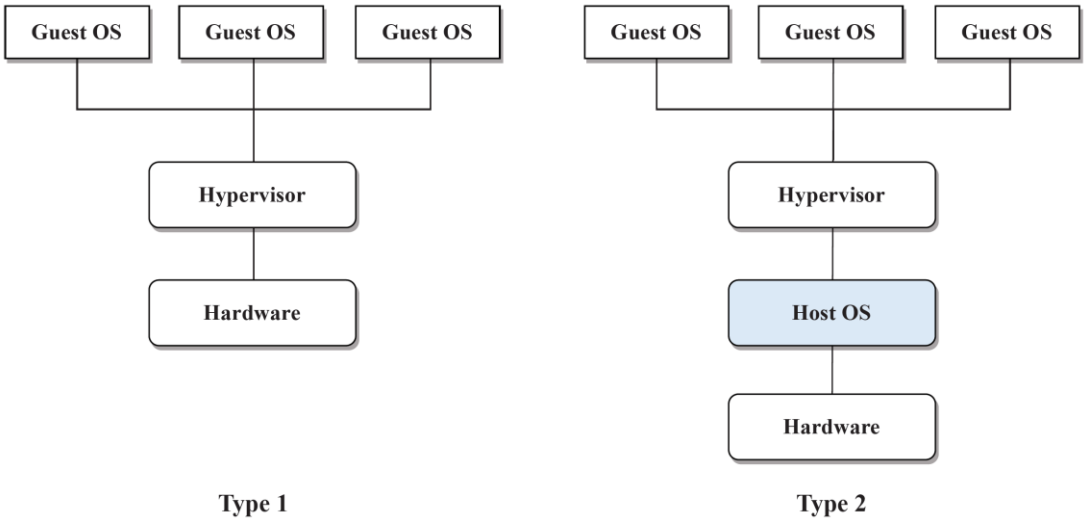

Figure 6: Hypervisor types.

In principle, there are two main approaches to hardware virtualization, which resulted in the two basic types of hypervisors. The Type-1 or the so-called bare-metal hypervisor runs directly on the physical machine and contains its own device drivers. The most prominent example of this hypervisor is XEN [65], and it operates independently as a unique, lightweight operating system. On the other hand, the Type-2 or the so-called hosted hypervisor requires the host OS's existence and does not contain any device drivers since it runs under the standard operating system. The essential characteristic of a hosted hypervisor is that the guest OS runs as a process on a host OS, and it is therefore handled by the host scheduler like any other process. The advantage of this approach is improved security and

system scalability because VMs could run unmodified guest OSs. The typical example for a Type-2 hypervisor is KVM, which is a loadable module to the Linux kernel. However, the latest editions of the KVM have specific drivers such as libvirt, which enable efficient guest OS access to the physical peripherals. In practice, there is usually only one hypervisor per host OS that manages all virtual machines, although specific implementations such as vSphere of VMware could attach one VMM (virtual machine manager) to each VM. This example could create slight confusion as the term VMM is typically used interchangeably with the term hypervisor. In this thesis, the term VMM will be used exclusively as the substitute for the term hypervisor.

3.3.4 QEMU

QEMU is an open-source computer hardware emulator that uses dynamic binary translation for improving performance during instruction set architecture (ISA) execution. Because virtualization implies two OSs, one in the host and the other in the guest machine, QEMU executes one ISA on top of another. It enables running multiple unmodified guest OSs by supporting various hardware and device model implementations. Generally, ISA development is critical in computer hardware design because it ensures binary compatibility between different generations of computers. Besides x86, QEMU also supports other architectures, such as MIPS, SPARC, and PowerPC, to name a few. In addition to hardware compatibility, QEMU is very versatile regarding software support, and it can operate under most modern OSs such as Windows, MAC OS, and Linux. Developers are particularly interested in using QEMU, mostly when they work on rare or not easy available systems. QEMU also plays a vital role in software debugging because it enables running various guest OSs without requiring a system reboot. However, the most important role of QEMU is virtualization and user-lever hypervisor support that is built-in into its code.

In general, QEMU runs in two main modes of operation: a user-mode emulation and a full system emulation. The user-mode emulation refers to the applications compiled on one ISA that can run on another by converting system calls. It also provides accurate signal handling by doing remapping between host and target signals. In addition, user-mode emulation enables Linux thread scheduling by

establishing access to the native CPU clone() calls. On the other hand, full system emulation allows the creation of virtual CPUs (vCPUs), virtual RAM, Discs, and devices to execute unmodified guest OS. It uses a full memory management unit (MMU) to achieve maximum portability and efficiency.

With the development of virtualization hardware accelerators such as Intel's VT-x, QEMU was able to work in parallel with the in-kernel module of KVM. As a consequence, KVM could execute most of the guest instructions at the native speeds. This development allowed for massive computational performance upgrades that were a necessary precondition for efficient cloud computing. Another essential feature of QEMU is support for symmetric multiprocessing (SMP), which became the base of parallelization by distributing host tasks between multiple logical processing units. However, SMP could not be used by the guest OS if KVM is not active. Therefore, it is clear that both QEMU and KVM have to be engaged simultaneously if the goal of virtualization is high-performance computing.

However, the typical guest OS runs notably slower than its host counterpart because the binary translation creates considerable emulation overheads. This inefficiency derives from emulator requirement for high versatility and multi-architecture support, resulting in insufficient platform optimization.

3.3.5 KVM

Kernel-based virtual machine (KVM) is an open-source project that started in 2006 by creating an efficient Linux hypervisor designed for x86 architecture. KVM consists of two main components that have different tasks based on address-space access privileges. The first component contains two loadable kernel modules, kvm.ko and kvm-intel.ko (kvm-amd.ko), which are both included in the mainline Linux as of version 2.6.20. The kvm.ko provides a multi-architecture solution for virtualization, while the processor-specific modules, kvm-intel.ko (kvm-amd.ko) allow hardware acceleration based on virtualization extensions (Intel-T or AMD-V). The introduction of kernel modules emerged from the privileged-mode execution requirement of new x86 virtualization instructions, which significantly improved virtualization efficiency.

The second user-space component, QEMU, allowed hardware emulation, such as the creation of virtual CPUs, memory, and network cards. Together, they constitute a QEMU/KVM system, a term for Linux hypervisor in a broad sense. However, the abbreviation KVM is often used to describe the hypervisor itself, although it is clear that the name refers to the whole project and not the kernel modules only.

Figure 7 depicts the QEMU/KVM hypervisor and transition from the user-space QEMU process, which emulates the VM hardware, to the kernel KVM module. The communication occurs via the /dev/kvm device file created after loading the kvm.ko module. With this implementation, instead of standard CPU hardware emulation, QEMU entrusts the KVM (kernel) module to run x86 virtualization instructions directly on the host processor. In return, QEMU performs memory allocation and virtual device creation, which will operate as guest's virtual RAM and user device, respectively. Consequently, shifting the CPU emulation to the kernel module allowed for significant performance improvements and established virtual machines as the perspective computing platforms.

Although virtual machines are typically created with dedicated VM managers, such as virsh-install, virt-Manager, or OpenStack, QEMU/KVM also allows VM setup with the command-line interface. For instance, only a few input parameters are required for the basic VM creation:

qemu-kvm –smp 2 –hda ubuntu.img -m 2G

The example above is a very simple showcase for launching a VM. By running the qemu-kvm executable with only three parameters, it is possible to create a VM with Ubuntu guest OS, 2 vCPUs (smp), and 2 GBs of RAM. Hoverer, the typical QEMU/KVM input set for the OpenStack VM creation contains a list of several dozen parameters.

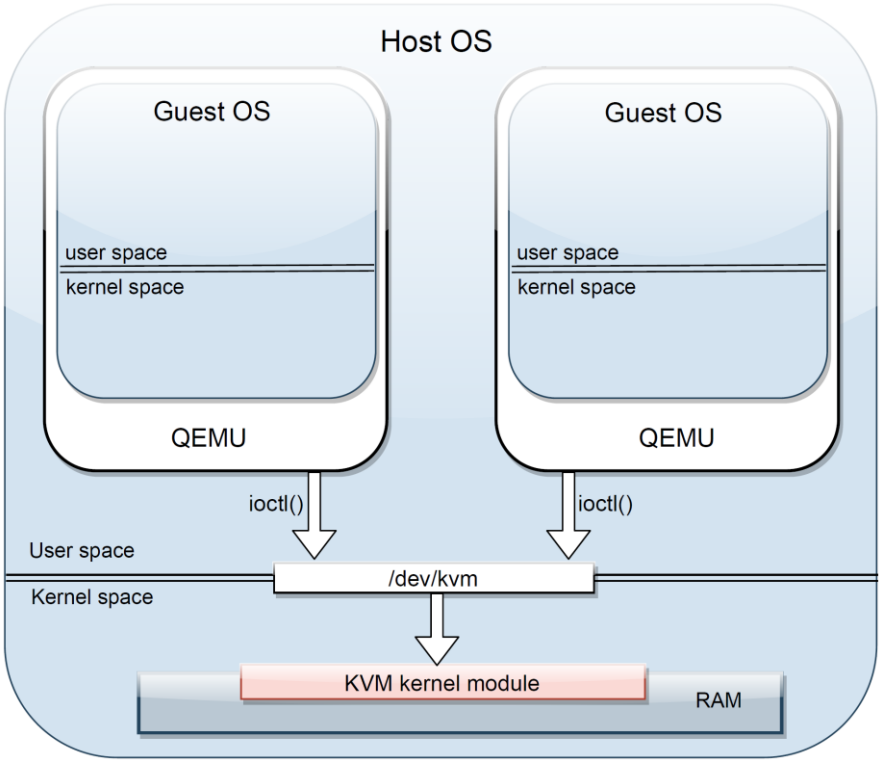

Figure 7: QEMU/KVM hypervisor.

Besides those mentioned above, OpenStack parameters include network devices, emulator, display, mouse, keyboard, multiple interfaces, metadata, and many other propriety settings. Also, it should be noted that OpenStack is a very complex cloud OS, and there are many dependencies, access permissions, and auxiliary software solutions that prevent the bypassing of Nova in the instance creation. In other words, starting a new OpenStack instance by using qemu-kvm executable from the example above is not allowed by the system.

Eventually, the KVM design provided a useful and flexible virtualization solution by building a hypervisor on top of the Linux kernel. Because operating systems and hypervisors share essentially very similar tasks such as process creation, scheduling, memory management, or device provision, developers managed to save significant time by applying already established software solutions. This approach turned the QEMU/KVM virtual machines into standard Linux processes that could be scheduled or managed like any other host process. As a result, many scheduling and optimization techniques such as vCPU-pinning and NUMA-aware scheduling could be later applied and improve VM applications efficiency.

3.3.6 Shared Memory in Virtual Machines

Ever since the introduction of virtualization, huge efforts have been made in improving the efficiency of execution and communication in a virtual environment. The first platform iterations, referred to as full-virtualization, produced enormous overheads, limiting VM adoption and broader usage. However, the situation dramatically changed with the introduction of paravirtualized drivers that allowed skipping of expensive hypervisor interceptions and more efficient real hardware access.

One of the pioneers of paravirtualized driver development and standardization, Rusty Russel [66], focused his efforts on I/O virtual devices and the virtio mechanism creation. Russel explained his motivation in the following way: bare-metal devices have very fast access to device registers while virtual devices, due to hypervisor interception, access their I/O registers rather slow. On the other hand, memory access from virtual devices is fast and direct, while real devices require the engagement of time-consuming DMA operations. Therefore, by creating a hybrid, paravirtualized solution, virtio would use the best of both worlds and thus enable the ultimate virtual driver performance. After standardization, virtio included a family of PCI devices, although Linux implementations allowed shared, non-PCI drivers as well.

This concept of using virtual PCI devices for memory access is further explored by Macdonell in his Ph.D. thesis [11] when he presented the so-called Nahanni mechanism for shared memory communication. In his work, Macdonell introduced the ivshmem, a virtual PCI device whose registers were mapped to the host's main memory, allowing a direct guest-to-host or guest-to-guest data exchange. He also created a small host tool for shared memory access synchronization called SMS and even announced the creation of a simple MPI communication channel. However, this MPI channel was never released or published, and its effectiveness could not be tested nor confirmed. Meanwhile, regular Linux updates caused the corruption of SMS, and the whole concept of inter-VM shared memory communication became utterly flawed. Later, another research group [67] attempted to resurrect the ivshmem-based MPI channel and even contacted Macdonell for assistance, but this project was eventually abandoned and its online repository removed.

On the other hand, ivshmem, as a virtual PCI device, became part of an official QEMU release and, therefore, available for public access. Also, the team of researchers [68] further explored ivshmem for inter-VM communication, although their solutions aimed primarily at specific use-cases that included overcommitting hardware resources, which was bad for efficient cloud simulation. Finally, in 2017 the Ohio State University released the first version of an MPI variant called MVAPICH2-virt [69], which promised ivshmem optimization for SR-IOV enabled hardware. Although this was a hardware/software-specific solution, with particular modifications, it was possible to adapt it for our OpenStack micromagnetic project and use it for the intra-server measurements.

3.4 Linux Kernel

Linux is an open-source clone of the Unix operating system that started as a student project by Linus Torvalds in 1991. However, although it kept the most prominent features of the original Unix OS, such as virtual memory, shared libraries, multitasking, and networking, the source files of the new Linux kernel remained freely obtainable and universally accessible. Therefore, by making development tools, runtime libraries, and device drivers available to other developers, an immense paradigm shift occurred that further accelerated the Linux enrolment in industry and the private sector.

This rapid success of Linux came as a result of support from the worldwide programming communities that regarded its kernel as a base for future open-source projects. Additionally, by making the Linux kernel compatible with the POSIX standard, further OS improvements included contributions from many public university and student projects, which help structuring the code development [70].

Although designed initially for 32-bit x86 architecture with no portability prospects, Linux today supports multiple ISAs in both 32/64 variants. Moreover, all of the current 500 fastest supercomputers worldwide (Jun 2020) run on some Linux distribution [71], which is an excellent confirmation of its software quality and efficiency.

One of the most prominent features of Linux is its handling and scheduling of the processes. Namely, the Linux kernel recognizes each task, process, or thread as an individual process with a unique process ID (PID), which could be separately scheduled onto the corresponding CPU. This concept is fundamental because threads have certain features different than processes, such as shared resources: registers and address space, but undergo the same scheduling rules as the standard process. This is why threads are sometimes called light-weight processes (LWP), which could be confusing as many developers and Linux tool authors use different terminologies. Also, one should not mix threads (LWP) with Hyperthreading [72], which is Intel's physical extension of existing CPU core capability aimed at multitasking improvement.

As a monolithic kernel, the Linux kernel supports the so-called preemptive scheduling. However, true preemptive multitasking became available only with version 2.6, which means that both user and kernel-mode execution processes could be preemptively interrupted. Previously, it was possible for the Linux kernel to de-schedule only the user-mode processes. However, compared with the other monolithic kernels, the Linux kernel allows loading the device drivers as modules, even during system runtime. Consequently, it was possible to interrupt even the device driver execution, which further improved multiprocessing and hardware interruption handling.

Generally, Linux device drivers run in the system (kernel) space, but there are some exceptions, such as UIO drivers. UIO drivers allow direct access from the user application to the kernel, which was previously reserved only for the system calls. Furthermore, operations on UIO device drivers could be used for the synchronization between virtual machines, which is a concept that will be explored later in chapter 6.2.1.

Finally, Linux offers many security mechanisms that are commonly known as Linux security modules (LSM). The most famous LSMs are SELinux [73] and AppArmor [74], and both of them are supporting a range of different security policies, such as mandatory access control (MAC) [75], for instance. On the other hand, braking VM isolation to enhance the inter-VM intra-host communication requires the deactivation of most LSM layers, which is often complicated and time-consuming. However, investing additional time in enabling shared memory access for multiple

intra-host VMs is fully justified with performance gains achieved by enhanced SHM communication.

3.5 Scheduling

The Scheduler is a software algorithm that allows efficient sharing of computer resources among multiple users by keeping the system load in balance. It provides the methods for assigning the work elements such as processes or threads onto the hardware resources. These activities are referred to as scheduling and allow for computer multitasking, even when hardware contains only a single CPU. However, the practical implementation of scheduling algorithms often requires compromises because of conflicting goals in the efficient multitasking system design. For instance, by maximizing the throughput of achieved work per time and minimizing the waiting time before execution, a conflict with the scheduling fairness or the balanced CPU time distribution is created. Depending on the user requirements, the scheduling priority is put on one of the goals mentioned above.

3.5.1 Process Scheduler

Process or CPU scheduler is the so-called short-term scheduler that decides the execution order of processes in dedicated time slots. It also determines the priority of CPU allocation upon reception of different signals, interruptions, or system calls. Based on the interruption capability, CPU schedulers are divided into two main categories: preemptive and non-preemptive (or co-operative). As the name suggests, the Preemptive scheduler can forcibly de-schedule the CPU process and replace it with another one by engaging the interrupt handler [76]. Contrary, the non-preemptive scheduler does not have this functionality and has to wait until the end of process execution.

The older Linux iterations, starting with the kernel version 2.6.0, used the so-called O(1) scheduler [77], which was developed by Ingo Molnar. The scheduler itself received multiple updates and patches afterward, out of which the most prominent was Con Kolivas' implementation of "fair scheduling". Inspired by Kolivas' work, Molnar later developed the first widely spread scheduler based on fair queuing and

name it the completely fair scheduler (CSF). Starting with Linux 2.6.23, CSF becomes the standard Linux process scheduler, which remained until today.

3.5.1.1 Completely Fair Scheduler (CFS)

Completely fair scheduler (CSF) is designed to increase the system performance by maximizing CPU utilization. The goal of the "fair scheduling" algorithm is that every process receives an equal share of processing power, which is the concept borrowed from the idea of "ideal processor". This is achieved by implementing a red-black tree algorithm instead of run-queues, present on the previous O(1) scheduler. Additionally, new atomic precision was introduced that allowed nanosecond granulation of CPU share that each process had on disposal. Consequently, the atomic-precision made obsolete the application of time slices employed by O(1), which improved the overall scheduling efficiency. Finally, CFS also encompassed the so-called "sleeping" processes or tasks waiting for user interaction and considered them equal to those on the active execution queue. Although this concept of "sleeper fairness" was taken from O(1), it allows for corresponding priority distribution to processes that spent most of the time in waiting-for-the-event or, alternatively, user input state. The illustration of the red-black tree algorithm is given in figure 8.

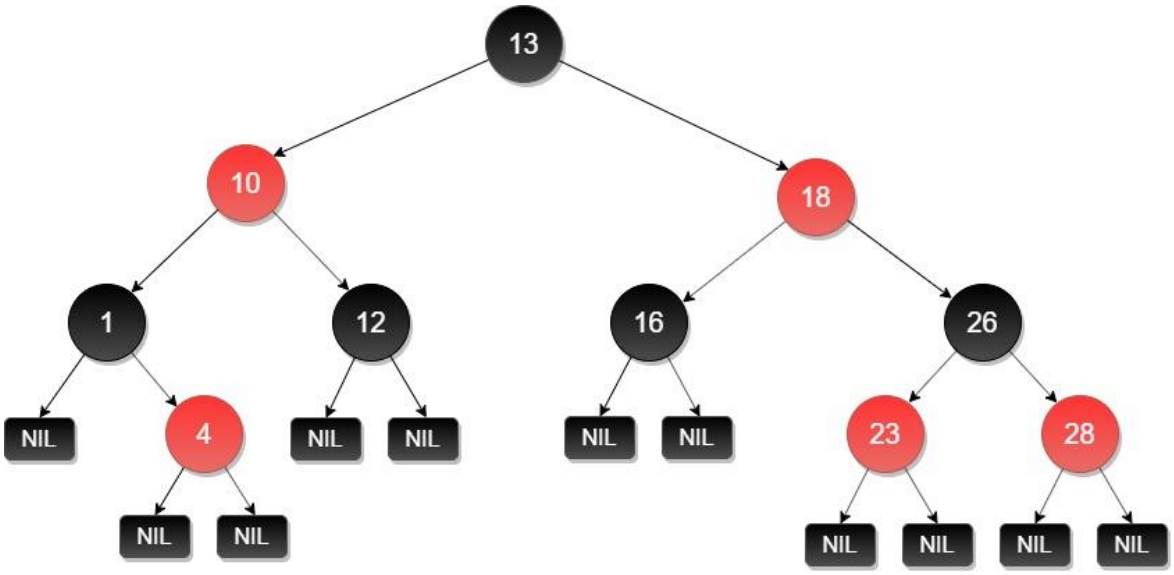

Figure 8: Red-black tree algorithm example.

Each node colored red or black contains the processor “execution time”, given in nanoseconds. Also, another value called “maximum execution time” is introduced and calculated in the following way: the process waiting time for execution is divided by the total number of the processes. This calculated value, therefore, corresponds to the expected runtime on the “ideal processor”. The scheduling occurs by the following rules:

1) The most left node, which has the lowest “execution time”, will always be the first scheduled from the red-black tree
2) The scheduling always goes from left to right
3) If the process is entirely executed before its maximum execution time expires, it is removed from the tree
4) If the maximum execution time expires or preemptive interruption occurs, the process is reinserted into the tree based on the new value of “execution time”

It is clear from the description above that the more time the process spends in the execution, the greater the value “execution time” becomes, and the process is shifted further to the right in the tree. This shifting explains the priority given to the waiting processes because their “execution time” is correspondingly low. However, the most prominent feature of the red-black tree algorithm is described as following: The path from the root to the farthest leaf (NIL) could be maximally two times longer than one from the root to the closest leaf. This important characteristic of the red-black tree brings in the balanced height of the tree, which significantly reduces the search steps for each of the n scheduling nodes.

Finally, the development of CFS did not stop after the initial release. Later patches, introduced at first in 2010 and later in 2016, further improved scheduling efficiency and multicore capabilities. However, as mentioned earlier, scheduling efficiency directly depends on compromises and application goals. Sometimes the general-purpose solutions such as the “fair scheduling” algorithm are not correctly fitted for the HPC simulations, which is often noticeable in cloud applications. In the following chapter, this issue will be given in more detail as well as suggested solutions.

3.6 Inter-Process Communication

Inter-process communication (IPC) is an essential computing mechanism that allows for synchronized data exchange between processes. This data exchange is thus regarded as a method of cooperation between the processes. Based on the type of IPC, the data is typically divided into two main groups. The first data type, called “Shared data”, is significant to applications running on the same host because it enables simultaneous access of multiple processes to the common data set. Of course, the synchronization mechanism, in this case, plays a critical role as it prevents data corruption during shared-memory access. The second type of IPC data is called “Streamed data” and operates quite contrary by transferring the data entirely from one process to another. This method is needed for geographically dislocated or remote processes, which is characteristic of distributed systems. For instance, Stream data IPC occurs during the parallel run of cloud applications in the form of inter-server communication. However, it could also refer to some commercial service access, such as web browsing.

In general, operating systems support multiple IPCs to reflect the application’s communication requirements better. The standard and the most commonly used IPCs in Linux are pipes, sockets, signals, and shared memory, but other mechanisms exist as well, such as eventfd [78], for example.

Ultimately, the kind of IPC that will be used is determined by the application requirements. Also, it is often the case that several IPCs will be applied simultaneously to achieve the best performance for each communication scenario.

3.7 Message Passing Interface

Message passing interface (MPI) is an open-source communication protocol designed as an API for programming parallel computers. Besides high system scalability and platform portability, it enables the creation of high-performance computing applications by providing native optimization for the underlying hardware. These characteristics ensured that the vast majority of scientific or engineering work performed on supercomputers today is done with some version of MPI standard. As a result, MPI is practically unmatched when it comes to

competition in HPC, and it is implemented or supported by almost every distributed memory system in the world.

In principle, there are two major development branches of MPI: MPICH and Open MPI. Although they started as separate projects, today, both implementations build their systems on top of common MPI standards, which reached the stage MPI-3 at the moment of writing. The development practice established over time implied that each successor software version should be a superset of the previous one, while only a small set of functions were to be depreciated over time. This approach allows for the renewal of older applications and performance upgrades by replacing the obsolete MPI libraries with the newer versions. However, this process could become very complicated since it requires an in-depth knowledge of software structures and library organizations, mainly when there is a significant gap between MPI versions.

Traditionally, MPI belongs to the OSI layer 5, but the transport layers, such as TCP or sockets, have to be implemented as well. It also provides excellent compatibility with many development platforms by directly supporting C, C++, and Fortran programming languages. However, it is possible to combine MPI with Java or Python if shared libraries are engaged. When it comes to functionality, MPI enables virtual topology, synchronization, and communication between spawn processes. In order to achieve maximum performance, each logical processing unit of a host, which could be a CPU core or a thread, would be assigned a single MPI process. This assignment procedure will be performed in runtime by the agents called mpiexec or mpirun. In essence, these are the scripts responsible for initiating MPI parallel execution and constitute an integral part of a process manager.

Finally, MPI supports both point-to-point and collective communications, although its basic code requires the usage of only six functions.

3.7.1 MVAPICH2-virt

MVAPICH2-virt [69] is a free MPI-3 software package designed at the Ohio State University/USA. In contrast to many similar MPI iterations, its most prominent feature is the built-in support for ivshmem. However, because MVAPICH2-virt was

initially developed to support Infiniband networks in a virtual environment, its utilization requires modification of a standard installation process. It is also mandatory to follow the corresponding and detailed guidelines of the Mellanox Corp [27], which have to be carefully selected based on the exact hardware and software configuration.

The setup is accomplished by engaging the specific propriety APIs instead of standard OFED verbs [79], typically used in Linux as device drivers for the Infiniband. In addition, MVAPICH2-virt does not come with its source code, which removes flexibility in software optimization and the selection of different communication channels. Furthermore, it prevents insight into the underlying shared-memory synchronization mechanism since this information is not covered with available user documentation. Consequently, MVAPICH2-virt uses TCP/IP for the indispensable mutual exclusion of multiple processes when accessing the same shared memory region. As a result, this TCP/IP-based shared memory access-synchronization implies a higher latency for each access switch.

3.8 Device Drivers

Device drivers are computer codes that provide a software interface between the host OS and physical devices. Due to their functionality and efficiency, most modern operating systems such as MS Windows, Linux, or Mac OS use device drivers to control and operate their peripheral devices. Device drivers are designed modularly, bringing more stability in the operating system execution because they act as a translator between applications and peripheral devices. The communication between an application and hardware goes as follows: First, the calling program invokes the routine in the device driver, which translates it and sends further the corresponding commands to the physical device. Once the device completes the desired actions, it sends the data back to the device driver. Depending on the peripheral device or application requirement, device drivers may initiate routines in the original calling program. With this approach, programmers could develop hardware-independent applications because implementation details are encapsulated within the driver. On the other hand, device drivers are highly dependable on the operating system and the specific hardware that they operate.

To categorize the different device drivers, it is necessary first to define the different types of devices. Typically, there are three main categories of devices: character devices, network devices, and block devices. This means that each device driver needs to be initially identified either as a char (character), network, or a block device before device configuration is completed.

As their name suggests, the block devices organize their data in blocks that are fixed in size. Correspondingly, the data exchange between the block devices occurs by sending and receiving blocks of data equal in size. Because of the defined and fixed length of blocks, it is possible to access these drivers and their data randomly. The most prominent examples for the block devices are file system entries.

Compared with the block devices, character devices do not store the data but instead operate by allowing the so-called data streams. These streams of bytes are either sent or received between the devices, such as displays and serial ports, which in most cases, prevent random data access. One of the main features of a character device that directly affects virtualization is its support for POSIX system calls such as read(), write(), mmap(), or ioctl(). This inter-platform interaction was achieved by calling these functions directly from the guest user-space and executing them on the corresponding device file. The guest OS generates these device files within /dev system folder upon device installation. For instance, mmap() system call performed on corresponding /dev/ivshmem device file can map the register-address of ivshmem PCI device directly on the host's main memory, thus allowing a shared memory between multiple VMs. As a result, VMs can share the data directly and without buffering, which is the most efficient data exchange method.

Another example of char POSIX support is the execution of the read() function on the device file, which causes the blocking of a user code execution. The blocking read is one of the design features of the UIO device and can be used to create the custom mechanism for shared memory access synchronization. However, the unblocking requires system interruption, which generally could not occur between VMs as each VM has its own guest OS. This is a case where the "mmap-ing" of ivshmem device registers plays a vital role as it allows QEMU intervention in host OS. The details of this signaling mechanism will be presented in chapter 6.2.1.

3.9 Hardware Layout

Since the introduction of paravirtualization and a widespread application of binary translation, the number of virtual machine users has grown exponentially. This development came as no surprise as mentioned technologies resulted in more efficient virtualization of x86 architecture. Driven by the popularity of virtualization, the two biggest x86 ISA manufacturers, Intel and AMD, decided to improve the existing hypervisor efficiency and introduce new propriety hardware extensions. Their initial goal was to remove binary translation because of its performance limitations while reducing virtualization's adverse effects by engaging a paravirtualized approach.

This upgrade was achieved by introducing the new instruction set that would simplify the creation, management, and execution of VMs. The first solution came in late 2005 from Intel, and it was called virtual machine extensions or short VMX. AMD did not wait long after and, in the following year, released its product that was called secure virtual machine or SVM. However, these two hardware extensions were not compatible with each other because of design differences and the lack of standardization. To this day, these two technologies remained the industry standards for CPU and memory virtualization, although other accelerators such as I/O extensions Vt-d [80] or SR-IOV [61] were introduced as well.

Following the initial release of both VMX and SVM extensions, the software market quickly responded and exploited the new hardware accelerator capabilities. One of the first hypervisors that emerged in the same year (2006) was Parallels [81], which was quickly followed by the Xen [65] release. Finally, in the same year, the KVM project started and quickly became the standard Linux hypervisor.

3.9.1 VT-x: Intel Virtualization Technology

VT-x is an integral concept of Intel's virtualization effort for x86 architecture, aiming the efficient computation and VM execution on the underlying hardware. The desired computation efficiency is achieved by introducing new administrative hardware that is generally needed for virtualization and lowering it to a non-virtualized server. This concept enables the offloading of a hypervisor as the

interceptor and allows direct execution of guest instruction set on the host hardware. VT-x consists of three unique technologies: virtual machine extension (VMX), virtual machine control structure (VMCS), and extended page tables (EPT).

3.9.1.1 Virtual Machine Extension (VMX)

VMX consists of virtual machine extensions that are introduced into the command set of the CPU. The new commands, such as VMLAUNCH, VMXON, VMRESUME, and VMXOFF, allowed for more efficient starting, managing, and terminating virtual machines than the old, software-based IA32 commands. Also, VMX operates in close connection with the virtual machine control structure (VMCS), and together they improve the hardware virtualization performance while increasing the overall system safety.

VMX differentiates two sets of commands based on access privileges: commands with root privileges running in kernel mode and others with non-root privileges, which operate in user-mode. Therefore, all root commands could only be executed by the host OS or hypervisor, while the guest OS could call the not-root ones. Besides, CPUs do not contain extra status bits that differentiate particular access modes, which means that guest OS is entirely unaware of underlying virtualization. Finally, there are two terms reserved for the transition between the two privileged modes. The first is called “VM entry”, and it designates the transition from the root to the non-root mode. The second mode is called “VM exit”. It describes the return from non-root to root privilege mode. These transitions are essential, as the proper privilege mode determines the VM execution efficiency.

4 SOLVER SELECTION

Abstract

In order to estimate the efficiency of simulation packages in a cloud environment, a series of extensive performance tests have to be conducted first. The initial project goal was to port and execute Magpar, Nmag, and Vampire in our private cloud and compare them in all relevant communication scenarios. As a result, the best candidate for software improvement should be selected and further enhanced. However, viable tests require a comprehensive set of input and output parameters designed from the bottom up. This setup implied: 1) creation of the unique input geometry with identical mesh size, shape, and the number of grid points and 2) using the same magnetic material with corresponding parameters, initial and boundary conditions, and the direction/intensity of the external write-field. Furthermore, all simulation parameters, such as numerical methods for time step integration, the order of discretization, number, and duration of time steps, have to be perfectly matched to yield reliable results. Last but not least, the output of the simulation has to be in the same format, and all additional features that are not present in other solvers switched off.

Consequently, the idea of simple software performance comparison became a challenge, particularly with poor solver documentation. Each of the packages mentioned above was developed without a standardized API and with completely different user interfaces and initial settings. For instance, the default input file for Nmag contained only half a dozen parameters while Magpar allowed for more than a hundred.

Since the magnetization vector time development was the primary output parameter, the fourth solver, OOMMF, was added as a reference for the other three. Once the proper input configuration was established, MPI-free OOMMF was no longer required. Finally, the tests were conducted with the highest efficiency in mind, and the overcommitment of physical resources was strictly avoided. This means that the number of launched parallel processes was never higher than the number of available physical cores.

4.1 **Input Configuration**

4.1.1 Material and Geometry Selection

The first step in establishing proper solver configuration was the magnetic material selection. We have decided to use the ferromagnetic magnetic material called Permalloy because it exhibits a high magnetic permeability and negligible magnetostriction [82], which is the main reason for its widespread industry application. Low magnetostriction of material means that it will not deform or change its size, shape, or other physical properties under the external magnetic field influence. As its name suggests, Permalloy is an alloy of nickel and iron with a ratio of approx. 80:20 respectively. One of the most important parameters of each magnetic material is its magnetic saturation, which is 0.86e6 A/m for Permalloy. The saturation presents the state of magnetic material, beyond which a further increase of the effective magnetic field Heff could not increase the magnetization.

Permalloy's other relevant parameters are exchange-coupling constant A=13.0e-12 J/m, dimensionless damping constant alfa=0.5, and the (selected) direction of the initial magnetization vector: x=1, y=0, z=1. The initial vector value is then normalized to a unity length by each solver, resulting in (x,y,z)=(0.707,0,0.707).

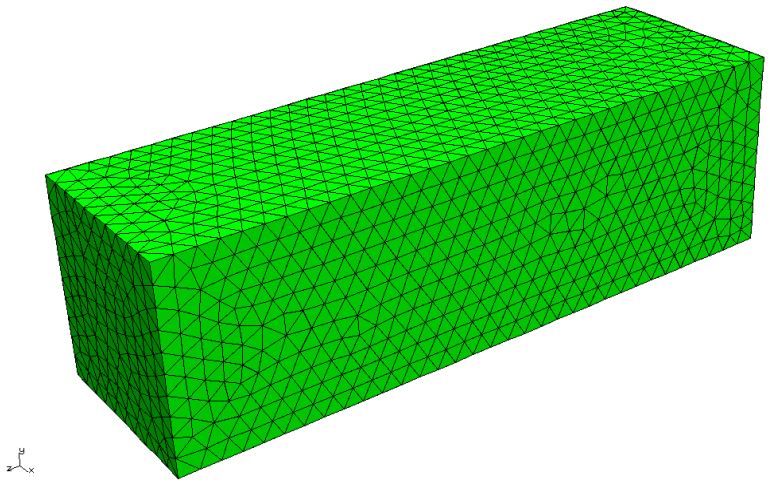

Figure 9: Input Mesh of the Permalloy bar.

The next step for the setup was the simulation system size definition or the geometry selection. Since Magpar and Nmag require external finite-element mesh generation, we have used the Netgen tool [44] to design the input shape. Due to known Permalloy properties and magnetization vector time/space development, the cuboid with dimensions (x,y,z)=(30,30,100) [nm] is selected. The resulting mesh is depicted in figure 9 and consists of 18,780 tetrahedral inside a bar, 4,100 grid points, and 3,460 surface elements. These parameters are essential as they significantly influence the simulation execution performance. During simulation initialization, the given mesh would be decomposed into pieces of approximately equal size, depending on the number of parallel processes engaged.

Consequently, each piece would be allocated to a unique logical processing unit for computation, which is a vCPU process in our cloud. Finally, OOMMF and Vampire are using a built-in geometry generation mechanism that simplifies the simulation setup. However, Vampire requires additional atomic crystal lettuce selection, which significantly increases computational load.

4.1.2 Simulation Input Configuration

The next step in the input configuration was the simulation parameters, such as the calculation time course. This value directly relates to the magnetization vector saturation, which is less than 300 picoseconds for the given geometry and ferromagnetic material (Permalloy). This time course is a function of the external write-field Heff that demagnetizes the initial magnetization value progressively. Therefore, we selected 300 [ps] as a fixed value for all simulations, as depicted in figure 10.

The second important parameter for proper solver comparison is the number of time steps for the numerical calculation. After several test runs, the optimal value of 6000 time-steps is determined to give the best output resolution. However, this estimate applies to all solvers except Vampire. The Vampire is designed to offer atomic precision, and this apparently low value for the time steps provided heavily distorted output (figure 11). It is empirically discovered that the optimal value for Vampire’s number of time steps is 120,000, or twenty times more. This

discrepancy indicates that Vampire is not suitable for further optimization, although it provides significant insights into atomic structures and physical effects.

Finally, the Newtonian numerical method is selected for all simulations, as it provides excellent results within dynamic time-integration. The equation's discretization order is set to level 2 since it allows for an optimal balance between performance and precision.

4.1.3 Output Results

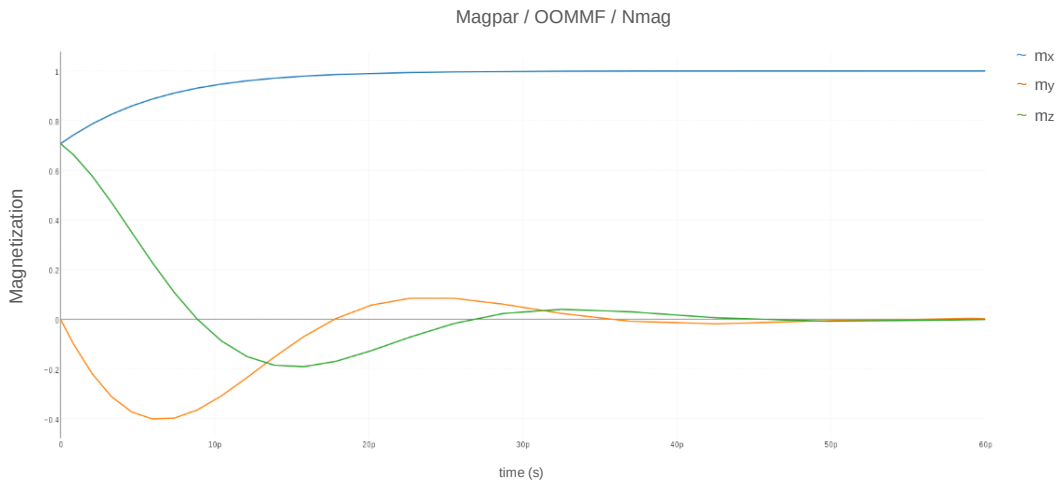

Figure 10: Time course of R^3 components of the magnetization vector M.

The final step in the configuration setup was the output plotting and data comparison. It was expected for a reliable comparison that the time development of each magnetization vector should be identical for all four solvers. As shown in figure 10, all simulation packages delivered the same result, which is remarkable considering substantial differences in solver design and input configuration. Figure 10 also depicts the z-axis alignment of the resulting magnetization vector (green curve) under the external field (Heff) influence. This alignment occurs at the moment of saturation when x and y components (blue/orange curves) of the magnetization vector are reaching zero.

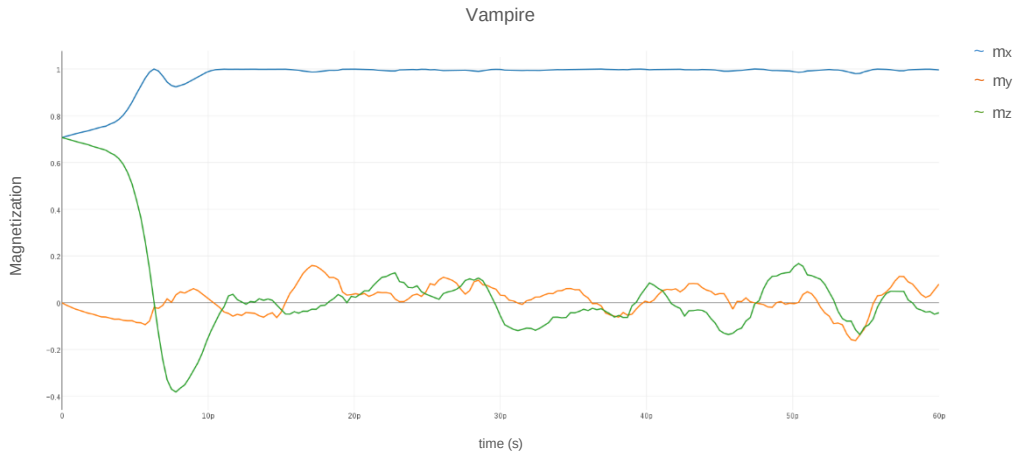

Figure 11: Vampire, magnetisation vector for time-step = 5 [ps]

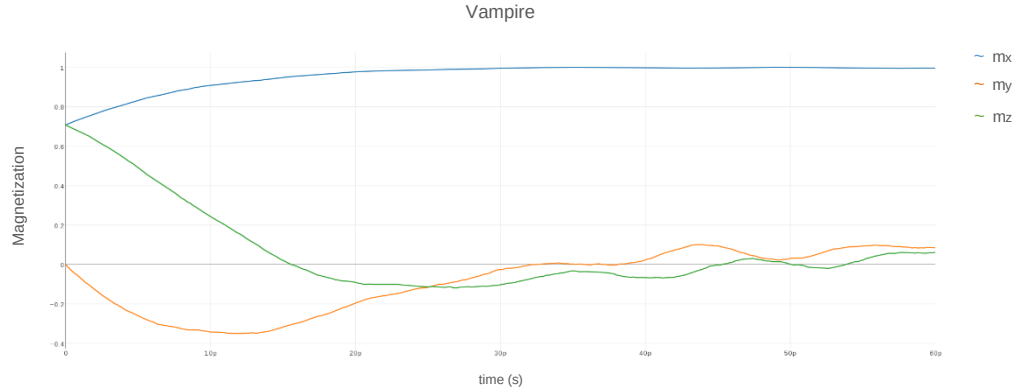

Figure 12: Vampire, magnetisation vector for time-step = 50 [fs]

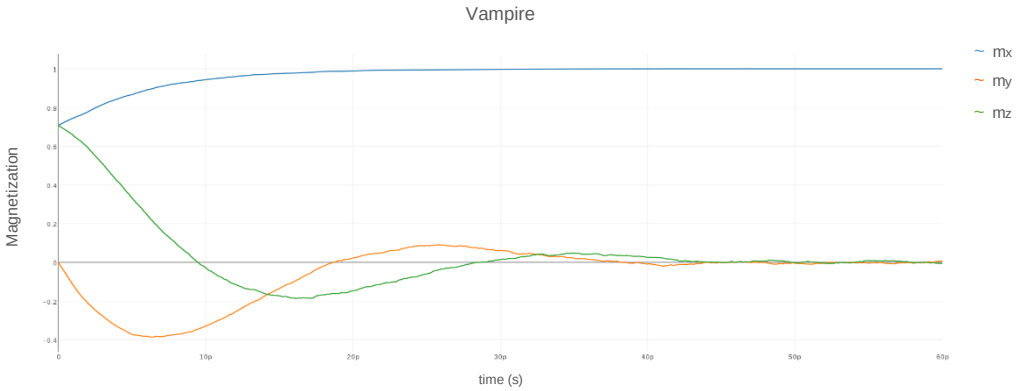

Figure 13: Vampire, magnetisation vector for time-step = 5 [fs]

However, to produce the same accurate results, Vampire needed days instead of minutes, compared with other solvers. The reason for this substantial computational load is the very high time-step resolution required for the atomic-scale simulation. Therefore, Vampire is excluded from further investigation, and the direct comparison is continued between Magpar and Nmag. The results of this comparison are presented in the next chapter.

4.2 Solver Comparison

In order to quantify the communication efficiency between parallel processes (vCPUs), virtual machines, and servers in a virtual environment, a comprehensive set of tests is used to exploit each IPC scenario maximally. Therefore, we have tested intra-VM, inter-VM, and inter-server communications as three primary IPC layers. In addition, the fourth setup was introduced to highlight the importance of the system size and its influence on the simulation performance. All tests comprised elapsed time (T) measurements of Magpar and Nmag in two of our most powerful servers. The basic setup included a Linux host server, QEMU/KVM hypervisor, and OpenStack as a cloud operating system. Initially, both solvers were executed sequentially as a reference for further parallelization efficiency estimation. In other words, each guest OS parallel process had a corresponding host OS equivalent, which is the vCPU light-weight process.

As shown in figure 9, individual vCPUs were responsible for a different mesh section since this system segmentation is performed at the solver initialization. The parallelization is achieved by engaging MPI, which was linked as an external library. Consequently, MPI did not function independently as a separate process, but it operated as an integral part of a solver. Finally, we used the so-called completely fair scheduler (CFS) [83] of Linux for our tests, which is one of 6 process schedulers available in the kernel. Therefore, the overall setup corresponded to the standard OpenStack cloud configuration, commonly seen in research institutions.

4.2.1 Scenario 1: Intra-VM Communication

The purpose of scenario 1 was to test the communication within the same virtual machine. This setup refers to inter-vCPU communication since each VM is statically allocated to one physical CPU by the OpenStack scheduler. Furthermore, the number of VM's parallel processes (vCPUs) was altered between 1 and 4. This interchange was applied to avoid overcommitting of resources since not more than 4 cores were available per physical CPU. As a result, the used vCPU variation allowed a 1-to-1 allocation of cores to vCPUs.

The measurement results are given in Table 1. It depicts the elapsed time T for both Magpar and Nmag, together with the simulation speedup S and the efficiency E. The two extra parameters, S and E, were added later in a post-processing phase: Speedup S is defined as the execution time on n parallel processes (vCPUs) compared to sequential execution (single vCPU). Efficiency E is then calculated as E=S/n, and it describes the vCPU utilization.

The first line of Table 1 shows elapsed time, speedup, and efficiency of sequential execution (1 vCPU) of solver in the guest OS. In this case, the VM translated the process generated by the solver into only 1 vCPU. On the other hand, the host's physical CPU executed two processes: the first process generated by the emulator (QEMU) presenting the VM in general, and the second process, vCPU, which is CPU emulation. The vCPU was forked-off as a VM child process and treated in the host as a light-weight process or a thread. This thread had its unique host process ID (PID), which means it could be scheduled by the host (Linux) scheduler. Also, each vCPU has the parent process ID or PPID, which is identical to the PID of the originating VM process. The given results in Table 1 for the sequential execution (1 vCPU) show that Magpar outperformed Nmag by a factor of 2.7 regarding the elapsed time T. This initial measurement outcome indicated a significant performance difference, considering the same input and output configuration. Additionally, the first line served as a reference for the subsequent parallel measurements, e.g., lines with 2-4 vCPUs.

Table 1: Elapsed time T, speedup S, and efficiency E for Magpar and Nmag, which were executed on 1 VM with n vCPUs. The n is varied from 1 to 4.

	Magpar			Nmag		
Setup	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**
1 vCPU	114	1	100	303	1	100
2 vCPUs	79	1.44	72	249	1.22	61
3 vCPUs	66	1.72	57.3	248	1.22	41
4 vCPUs	65	1.75	43.8	239	1.27	31.7

Parallel measurements also followed the same trend, described by 1 vCPU solver execution. As can be seen in Table 1, Magpar always scaled better than Nmag with respect to the speedup (S) and efficiency (E). However, speedups in both solvers never reached the theoretical maximum, which should be equal to the number of parallel processes. For instance, the maximum speedup value for Magpar is 1.75, which is far from the ideal value of 4, although 4 physical cores were engaged. Because our server employed an L3 cache configuration of the physical CPU, our initial expectations foresaw significantly higher speedup values.

However, in-depth analyses revealed the reasons for this sub-optimal performance of micromagnetic solvers. First, it was the influence of software overhead generated during virtualization. Although it did not have the same impact as the inter-VM communication overhead, it still reflects the performance costs of cloud deployment. Second, system segmentation and solver initialization could not be parallelized since these processes occur before IPC channel establishment. This initialization will undoubtedly impact each solver run, particularly those with relatively smaller system size, such as the mesh depicted in figure 9. Third, the implementation of shared memory communication in the virtual environment becomes more problematic as it requires occasional hypervisor intervention. As a result, direct shared memory access from guest OS does not occur, at least not in a common-sense of a non-virtualized environment. Fourth, different MPI implementations have unique settings for the default configuration of communication channels. Depending on the version, MPI libraries could support sockets, shared memory, or even auto-selection of available IPC channels.

Consequently, each case could result in a slight variation of the results and influence the measurement consistency. Fortunately, this scenario can be easily avoided by forcing the desired communication method during each MPI run. Finally, OpenStack is unaware of underlying physical hardware configuration, and guest OS could see only a generic product of computer emulation. This generic VM setup could not, for instance, differentiate between L2 and L3 level cache in the host server, which could lead to the omission of optimization mechanisms that are built-in into MPI. This scenario particularly applies to shared memory communication.

Regardless of solver initialization and final data assembly that does not require parallelization, virtualization does play a prominent role in intra-VM communication. However, server virtualization could not impact solver execution as significantly as in other scenarios, which will be addressed in the next chapter.

4.2.2 Scenario 2: Inter-VM Communication

The goal of scenario 2 was to test the communication between 2 virtual machines located on the same server. This type of cloud deployment is quite common, particularly with modern hardware, since powerful servers could host multiple or even dozens of VMs. Also, the critical aspect of these tests was the host's main memory engagement and its communication potential with respect to VM isolation. In addition, the Nova scheduler distributed VMs to different physical CPUs, which excluded utilization of the L3 CPU cache present in our most powerful machines. Finally, all VMs were created identical regarding OpenStack flavor, file system, and solver setup, which was confirmed by the XML configuration file. The measurement results are presented in Table 2. However, there is a slight difference in the presentation of the results compared with Table 1. As one can see, the speedup and efficiency are not starting with 100% since the reference value T is taken from the first line of Table 1, showing single vCPU execution.

Table 2: The influence of inter-VM communication on T, S, and E. Two physical CPUs execute the two VMs on the strongest server. The number n per VM is varied from 1 to 4, resulting in 2- 8 vCPUs.

	Magpar			Nmag		
Setup	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**
2 vCPU	115	0.99	49.6	309	0.98	49
4 vCPUs	91	1.25	31.3	284	1.07	26.7
6 vCPUs	91	1.25	20.8	280	1.08	18
8 vCPUs	102	1.12	13.9	311	0.97	12.2

The quick comparison of intra-VM and inter-VM measurements reveals a sharp performance drop in 2 vCPUs solver execution. Instead of an increase in the simulation speedup with parallel execution, we achieved a value that is smaller than 1. Although a lower performance was expected due to separate CPU deployment of VMs (no L3 cache between them), the result was still surprising. In general, the fastest communication between processing elements exists between the level 2 or level 3 caches inside of a CPU and allows 16 bytes to be transferred in about 1 ns. Obviously, this could not be the case.

As one can see, Magpar outperformed Nmag again in all respects. However, neither one of them reached a minimal elapsed time (T) value, which was 65s with Magpar (Table 1). Since shared main memory allows 1 byte to be transferred in about 1 ns, which is still quite fast, it was clear that the host's main memory was newer engaged for the inter-VM communication. The reason for this lies in virtualization and OpenStack network implementation. This VM design feature will be described and analyzed in greater detail in chapters 5.2 and 6.1.

Furthermore, the lowest T value in Table 2 is achieved using 4 or 6 vCPUs, and not 8 vCPUs. Because of this behavior, the system is not scalable, not even to the 8 parallel processes. There are two main reasons for this scaling saturation. First, virtualization creates substantial overhead and prevents efficient IPC between VMs, particularly by preventing main-memory communication. Second, the proportion between computation time and communication becomes worse if the 4100 grid points of figure 9 are distributed onto 8 vCPUs. In other words, the

problem size is not big enough for 8 vCPUs. Accordingly, the speedup and efficiency are highest for the case of 4 and, to some extent 6 vCPUs, but not for 8.

Finally, S never reached its maximum value of 1.75 from Table 1. This means that the best speedup is obtained if VM execution occurs on the same physical cores, although OpenStack is unaware of the host's cache coupling. The reason for that result lies in the L3 cache independence of OpenStack. It quickly delivers in many (but not all) cases data to a vCPU, regardless of these data were obtained via virtual shared memory, which was not the case, or via vNIC, which was true. However, a cache memory could only be engaged within the same physical CPU and not between VMs deployed on different sockets. Additionally, inter-VM data exchange suffers from considerable communication overhead, which is why we had a low value for S in the 8 vCPUs scenario.

4.2.3 Scenario 3: Inter-Server Communication

The aim of scenario 3 was to test the influence of inter-server communication on two VMs, which were located on two different servers and coupled with 1 Gbit/s Ethernet. In that scenario, the number of vCPUs per VM varied between 1 to 2, resulting in 2 or 4 vCPUs engaged for measurement, respectively. Each VM was allocated to one physical CPU, and each vCPU was assigned to one core.

Please note that Nova's command-line interface has to be used for this setup to manually allocate VMs to particular servers because the Horizon dashboard does not support this feature. In this way, the default settings of the cloud scheduler could be bypassed. This action was required because Nova chooses a host server of VMs based on a predefined set of criteria, which are essentially randomization algorithms. Also, as a reference value for S and E, the single vCPU execution time from Table 1 was engaged. The results are presented in Table 3.

Table 3: The impact of the inter-VM communication inter-servers communication via 1 Gbit/s Ethernet cards. The number of vCPU per VM is varied from 1 to 2.

	Magpar			Nmag		
Setup	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**
2 vCPUs / 2 VMs / 2 Servers	176	0.65	32.4	389	0.78	38.9
4 vCPUs / 4 VMs / 2 Servers	152	0.75	18.7	362	0.84	20.9

As one can see, the elapsed time T for 2 vCPUs in Table 1 is better by factor 2.2 than the corresponding T in Table 3 (176s). The difference has to be reconciled because the inter-server communication is the slowest cloud data exchange layer, as it requires 1 ns for 1 bit (1Gbps) transmission speed. This theoretical throughput is at least 128 times slower than the maximum inter-core (intra-CPU) interconnection speed, which was engaged in scenario 1. From the given results, it is clear that the maximum throughput of a physical link is not the dominant factor influencing simulation execution time, although it has a significant impact.

As mentioned earlier, we have simulation initialization time, which includes loading hundreds of simulation parameters, mesh processing, system segmentation, and testing of parallel computation nodes (channels), which are independent of communication type. Also, a massive portion of execution time is spent in the computation phase, which could be regarded as constant for each server unit. In addition, different hardware specifications of cloud nodes could cause an uneven distribution of computation time and thus increase the overall execution time. Finally, we have the inter-server communication overhead that engulfs all communication types, from slowest to fastest. It includes complete server visualization, starting from vNIC in guest, hypervisor, host drivers, physical NIC, and physical link (Ethernet). Of course, in practice, one acknowledged message has to go four times through the complete stack from vNIC to NIC and two times through the physical link.

Consequently, the fastest communication type is slowed down by the highest reduction factor, while the slowest type (inter-server) only by the smallest. This phenomenon is another reason we could observe a distorted proportion between

fast and slow communication, presented in Table 1 and Table 3, respectively. All in all, it is clear that total software overhead has to be reduced in order to assure cloud-efficient simulation execution.

Finally, Nmag's performance is again worse than Magpar's in absolute values (T), although it showed a slight improvement in speedup and efficiency compared with its initial measurements. Because of these results, the focus of the thesis from now on will be exclusively on Magpar, which is the best open-source micromagnetic solver for parallel cloud execution.

4.2.4 Scenario 4: System Hard Scaling

Table 4: Influence of the problem size on T, S, and E. Scenario 4 is identical to scenario 3, but only Magpar was measured with a different number of grid points.

	1 vCPU / 1 VM / 1 Server			2 vCPUs / 2 VMs / 2 Servers			4 vCPUs / 2 VMs / 2 Servers		
# grid points	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**
~ 4,000	114	1	100	176	0.65	32.4	152	0,75	18.7
~ 40,000	2616	1	100	2309	1.13	56.5	1477	1,77	44.3
~ 50,000	3900	1	100	3372	1.16	58	2215	1,76	44
~ 100,000	9428	1	100	8035	1.17	58.5	6004	1,57	39.3

The inter-server communication scenario revealed the discouraging fact that the speedup greater than 1 could not be achieved for the given input and system size. In order to find the tipping point and the problem size that could exploit our original cloud setup, the input mesh was varied between 4,100, 40,000, 50,000, and 100,000 grid points. As a reference, single VM sequential execution on our most powerful server was used for each problem size. Furthermore, the first line in Table 4 was copied from Table 1 and Table 3 for a better overview and comparison. It also shows that the simulation system size of 4,100 grid points is too small for enrolment on two servers, at least for the untuned cloud. In order to produce speedup greater than 1 for 2 vCPUs setup, it was discovered empirically that the mesh should contain about ten times more grid points. However, this resulted in a

speedup that only slightly surpassed (S=1.13) the acceleration threshold. Additionally, a further increase of mesh size did produce higher S and E, but this increase was small, and the level from where it started was also low.

On the other hand, in the configuration of 4 vCPUs, speedup and efficiency are steadily decreasing with increasing problem size. As a result, the best 4 vCPUs S and E values are those for problems bigger than 4,100 but smaller than 40,000 grid points, which is not ideal. Nevertheless, it means that it pays out to distribute big problems on multiple servers, but it is not efficient to compute them simultaneously on multiple vCPUs of the same VM, at least from a particular problem size on. In general, all these measurements clearly showed the potential for improving cloud efficiency, which will be discussed in the next chapter.

Finally, Table 4 results emphasized that the T for executing the nonlinear differential equations (e.g., Eq. 1) does not increase linearly with the problem size but instead over-proportionally. This development means that big micromagnetic systems can be solved only by parallel calculation because they have a non-linear computing complexity.

4.3 Summary

This chapter showed that speedup and efficiency metrics achieved in the typical OpenStack cloud could not be sufficient for HPC. Also, investigated solvers revealed that scaling to many parallel processes (vCPUs) poses a significant challenge if unmodified could is engaged. On the other hand, the presented measurement scenarios fully illuminated communication weaknesses and potential improvement points. In the next chapter, the necessary changes that will enhance our private cloud’s execution efficiency are given.

5 METHODOLOGY

Abstract

Successful integration of simulation software into the cloud environment requires recognition and resolution of issues related to the novel computing platform. In the first section of this chapter, the significant challenges associated with cloud execution efficiency will be identified and described. Then, in the second section, the proposed solution will be presented and discussed.

The chapter starts with communication problems and the so-called 3-level hierarchy recognized during data transfer. The next major topic addresses cloud architecture and the considerable overhead generated during inter-VM communication. This is particularly emphasized if the data exchange occurs on the same host server. Finally, we have examined the impact of inter-server latency on the commination efficiency as well as the contemporary challenges of Infiniband integration.

The proposed solution begins with the simulation software rebuilt and suggests the latest MPI parallel libraries implementation. It also deals with the recent Linux kernel changes and their effects on the relatively old micromagnetic simulation package. Addressing the CPU load balancing and the methods for mitigation of non-optimized process scheduling follows as a second topic, directly related to computing efficiency. We proposed the introduction of ivshmem or inter-VM shared memory that could significantly improve communication efficiency. Besides load balancing and MPI channel renewal, process distribution planning is added to solve the random scheduling/de-scheduling patterns of process managers, producing adverse effects on inter-server communication. Finally, the mistakes of previous Infiniband integration are examined, and proper implementation is suggested.

5.1 Communication Issues and 3-Level Hierarchy

The hardware and network stack analyses of cloud servers described in the previous chapter revealed a 3-level hierarchy present during inter-process communication. This IPC abstraction is essential because it allows substantially different communication speeds, which heavily influence cloud execution efficiency. As already mentioned, the fastest data exchange exists between the cores of the same CPU. It allows the communication speed of 16 bytes per nanosecond by using shared level-2 or level-3 cache memory (figure 14). This communication in our cloud terminology is also referred to as intra-VM or sometimes inter-vCPU, while all vCPUs belong to the same virtual machine.

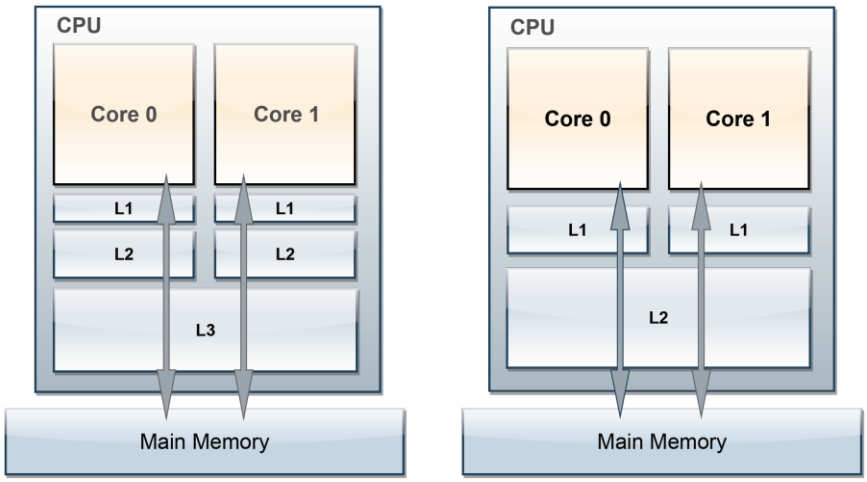

Figure 14: L3 and L2 CPU cache configuration.

The second-fastest communication within a server occurs between two CPUs, each inserted in its own CPU socket. This time, data exchange requires shared main memory since the cache does not exist between individual CPUs. Typically, the average data rate in this level-2 hierarchy IPC is a few times slower than the cache bandwidth, although the exact value depends on the data-set size, machine, or environment settings. However, VM isolation prevents cloud instances from accessing the host's shared memory and limits inter-VM communication efficiency. Finally, the slowest communication exists between the two servers and allows for a throughput of 1 bit per nanosecond only. Compared with the level-1 hierarchy, this transfer speed of inter-server communication is approximately 128 times

slower, which is significant. For instance, parallel computers and supercomputers have transfer speeds that are one or two orders of magnitude higher, respectively.

Therefore, it is essential that numerical algorithms take this into account in order to be cloud-efficient. In other words, new algorithms should enforce low inter-server communication while keeping at the same time high intra-server data rates. However, this approach has significant drawbacks. First, it requires rewriting all existing algorithms, which is an enormous task with uncertain benefits. Second, these new algorithms would still use the current communication infrastructure, which is inefficient for cloud execution, as shown in the previous chapter. Our approach suggests a more generalized solution that directly addresses week points of the present 3-level cloud hierarchy: inter-VM overhead and inter-VM communication speed.

The OpenStack communication stack analyses revealed a massive number of interfaces that are adding unnecessary overhead, lowering thus the overall cloud simulation performance. It is estimated that reducing particular elements could decrease the overall virtualization costs and enhance the micromagnetic solver execution. The second step is communication speed improvement, particularly within lever-2 and lever-3 hierarchy. More about these methods will be given in the following chapters.

Finally, besides enabling micromagnetic simulation enhancements, the suggested methods allow utilization in other research fields, such as solving different mathematical, physical, chemical, or biological problems. This project also highlights the importance of communication hierarchy recognition and its consideration in future simulation software development.

5.2 Inter-VM Intra-Server Communication Overhead

Once the solver comparison was completed and the initial analysis of measurement results accomplished, the next step was to identify the overhead produced by cloud virtualization. By studying the OpenStack literature [9] and examining our private cloud deployment, we have drawn a detailed cloud communication stack diagram (figure 15).

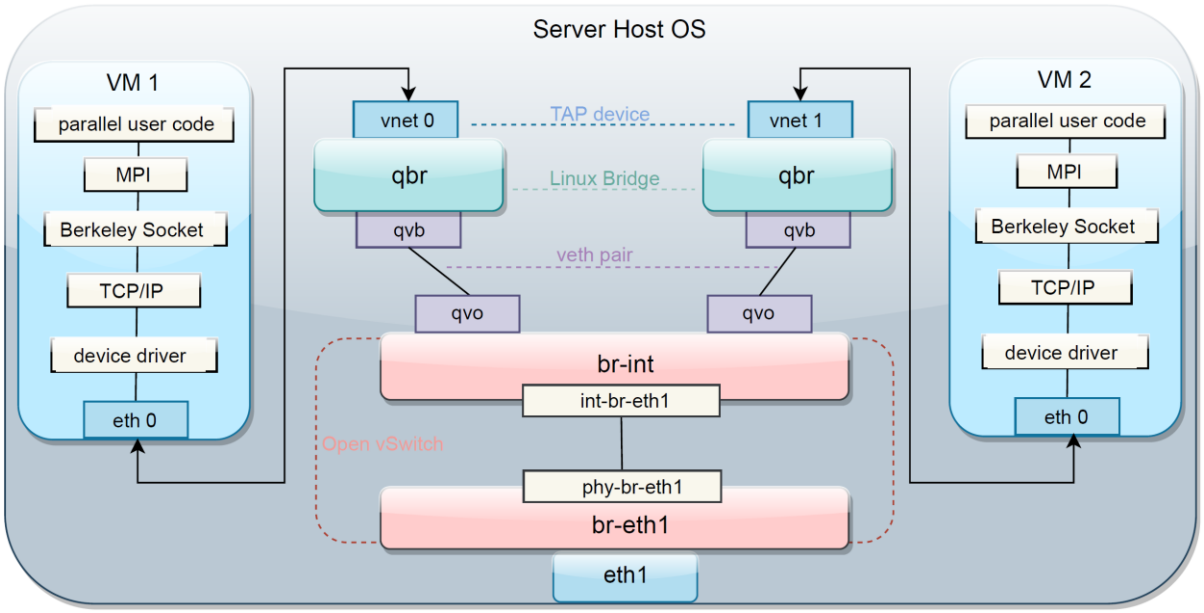

Figure 15: inter-VM intra-server communication in OpenStack.

However, a quick glimpse at figure 15 reveals the full complexity of cloud virtualization, depicting a vast number of interfaces engaged in inter-VM communication. Therefore, it is clear that the standard OpenStack implementation cannot be HPC efficient as each element introduces additional overhead during parallel processing or data exchange. On the other hand, the given solution enabled inexpensive virtual network deployment based exclusively on virtual or software-based devices. Also, the concept of VM isolation excluded unauthorized access from within the guest OS, which is useful if VM security is compromised, for instance. Nevertheless, this approach limits and, in many cases, prevents the efficient use of shared hardware resources, such as the host's main memory.

In order to illustrate the mentioned inter-VM communication challenges, we have created a scenario where two parallel processes calculate magnetization vector time development over the partitioned mesh. After each computing session, the parallel processes are application triggered to exchange intermediate results and then proceed with computations. The IPC initiated in the VM1 succeeds by sending data to the VM2 using the OpenStack communication stack. In other words, the inter-VM intra-server communication occurs by forwarding data through the following set of software interfaces:

Step 1: the data travels through the MPI process in VM1, Berkeley-Sockets, the TCP/IP stack, guest device driver, and virtual Network Interface Card (vNIC) - the final element within the VM1.

In step 2, the intermediate results go to the TAP device (vnet0), created by QEMU during VM initialization. TAP device operates on ISO Layer 2 and accepts the Ethernet frames generated by vNIC (eth0). The data is then forwarded to the Linux Bridge (qbr), which is also known as “Security Groups” in OpenStack. Although its primary role is OpenStack’s firewall configuration, qbr also acts as an interface between Open vSwitch (OVS) and VMs. This insertion was needed because OVS, as a third-party application, could not support a direct vNIC connection. The data then passes a pair of interfaces called “qvb” and “qvo”, also known as veth pairs. Veth pairs add address tag that is needed for the vNIC’s Ethernet frame in a virtual LAN (VLAN) environment. Finally, the tagged Ethernet frames reach the so-called Integration Bridge (br-int), which is part of the OVS. The Integration Bridge is essential as it makes the critical decision regarding the primary data branching. If the target VM is located outside the current server, br-int will combine the multiple VLANs into a single frame stream and forward it to the physical NIC.

In our case (step 3), the tagged Frames are sent back to the VM2, only this time in reverse. The tags are removed, and via the TAP device, the data ends up in vNIC of the VM2. Finally, as described in step 1, the data goes through the VM1 communication stack, only this time in the opposite direction.

From the given description, it is clear that this vast number of interfaces generates substantial overhead, particularly in dual side communication. Moreover, the real application would often involve multiple parallel processes with coinciding communication. Consequently, the standard cloud could not compare with a

supercomputer and not even a parallel computer. This project aims to change this and to improve cloud virtualization efficiency.

5.3 Inter-Server Overhead and Latency Limitations

The inter-VM communication between two servers becomes more complex than the intra-server data exchange for several reasons. First, the data has to be transferred to the physical NIC, which is a process that requires additional software interfaces. Second, the physical interconnection, i.e., Ethernet coupling, introduces latencies that are an order of magnitude higher than latencies within a single server. Furthermore, the data has to go a minimum of two times through the whole host/guest communication stack, which is hugely problematic.

The inter-server communication diagram is shown in figure 16. Also, to emphasize the OpenStack network segmentation, the diagram excludes the guest communication stack. As in the previous chapter, the same scenario where two parallel processes calculate magnetization vector time development over partitioned mesh is used. Maintaining the identical setup was necessary for obtaining comparable results. Finally, the data exchange path remains the same until the Open vSwitch: vNIC, TAP device, Linux Bridge (qbr), veth pairs, and Integration Bridge (br-int).

We have discussed that Integration Bridge is Neutron's central decision-making point, which is also responsible for combining multiple VLANs into a single frame stream. Once br-int recognizes that the target VM is located on the other server, it forwards the Ethernet frames from vNIC to the so-called Tunnel Bridge (br-eth1). These two Bridges are connected with another pair of interfaces (figure 16), and together they constitute the OVS. The central role of the Tunnel Bridge is packing/unpacking VLAN-tagged Ethernet Frames into real Ethernet Frames. In the L2 Neutron setup, Tunnel Bridge pushes the data through the host's device driver to the real NIC (eth1). Also, if both VMs are scheduled into the two separate compute nodes, there will be at least one physical switch between them.

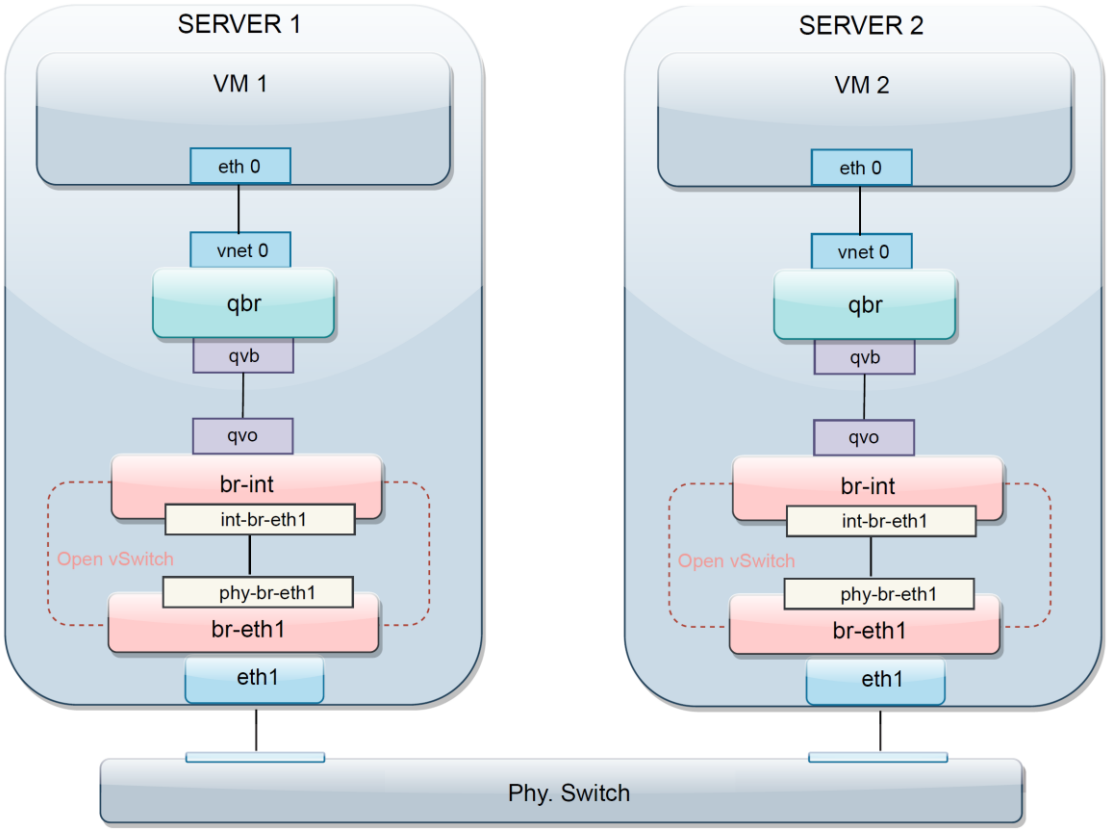

Figure 16: inter-VM inter-server communication in OpenStack.

However, compute nodes could share hardware resources with the central controller node, meaning that a router should be added to the diagram. Evidently, this common scenario in private clouds would further reduce communication efficiency. In the next step, the real Ethernet Frames from NIC output are sent to server 2 via the physical switch. Finally, the data needs to be transferred from the real NIC of server 2 to the guest application in VM2, this time in reverse.

The described inter-server communication introduces two additional steps in each server that were not present during the intra-server data exchange. First, the data is forwarded to the Tunnel Bridge through a pair of interfaces and then pushed via the device driver to the real NIC. In total, the inter-server scenario brings four extra steps of overhead without counting the interconnection latency. As mentioned earlier, 1Gbps Ethernet coupling is very poor for today's standards, and a successful upgrade should be a priority. However, implementation challenges need to be addressed first since OpenStack does not directly support novel

technologies, such as Infiniband. The next chapter will be dedicated to Infiniband integration challenges.

Finally, the current state-of-the-art in OpenStack could not result in a supercomputer replacement for simulation execution. Therefore, the proposed changes are expected to produce an efficient cloud execution environment.

5.3.1 Infiniband Integration Challenges

The standard cloud with Ethernet coupling exhibits poor interconnect capabilities, particularly regarding inter-server bandwidth and latency performance. Essentially, a cloud is a distributed system comprised of servers connected via LAN, meaning it could not compete even with a parallel computer. Furthermore, a simple communication system upgrade with Infiniband [27] or Myrinet [84] appeared to be insufficient, as the previous SimPass project [85] demonstrated. The main reason for this inefficient integration could be found in poor cooperation between the new hardware component and the OpenStack software interface. Also, standard Infiniband integration required a modified Tunnel Bridge, which induced extra communication overhead.

The entire network enhancement process involved several steps: First, the 1Gbps Ethernet NICs and Ethernet switch were replaced with 40Gbps Infiniband counterparts. Both cards and the switch belonged to hardware manufacturer Mellanox and were coupled with 1 m QSFP copper cables. The switch had small port-to-port latency, which was 100 nanoseconds. Second, an additional software interface was needed to modify and transfer data over Infiniband. This modification was achieved with a custom Linux TUN interface that encapsulated GRE [86] protocol as payload into an IP packet. These IP packets were then conveyed via IP-over-Infiniband (IPoIB) protocol [87] because Infiniband supports neither Ethernet Frames nor IP packets. Finally, original device drivers were replaced with OFED verbs [79], which serve as API for Infiniband.

However, the described Infiniband integration failed to deliver the expected results. The initial measurement revealed that the latency for smaller messages was worse than the original one (1Gbps Ethernet), which was disappointing. Also, the maximum data rate could not exceed 3Gbps, although Mellanox cards supported

up to 40Gbps. Finally, the main feature of Infiniband, the remote direct memory access (RDMA), could not be exploited because the available motherboards did not support hardware accelerators such as Vt-d [80] or SR-IOV [61]. It was clear that these issues have to be solved in order to turn the OpenStack cloud into an effective HPC platform. In the next chapter, the proposed methods for achieving these project goals will be presented.

5.4 Proposed Methods for Cloud Efficiency Improvement

5.4.1 Simulation Software Rebuild

Complex simulations are rarely attempted in the cloud because of huge software overheads and significant latencies in the virtual environment. It was also challenging to achieve noticeable performance gains in multi-VM setup, particularly when standard TCP/IP communication channels were engaged. As a result, even the simplest, two VM OpenStack setup could not justify cloud simulation deployment for many data exchange scenarios, as demonstrated in our measurements. In part, the reason for this inefficiency of communication libraries is a lack of continuous software development and maintenance once solvers reached a mature phase. In the case of Magpar, this support ended in 2010, which is quite a lot in terms of technology development. For instance, MPI parallel libraries received significant upgrades with respect to efficiency and usability during this period, while original ones became obsolete. Consequently, unused software ended up archived in online software repositories or, in some cases, even completely removed. Therefore, it was necessary to rebuild the Magpar with the latest MPI versions to maximize the parallel execution performance.

Of course, simple reconfiguration and compilation were not options as the latest MPI iterations, such as mpich3.2, received folder and subfolder structure changes besides new libraries. Also, a specific understanding of the Magpar package organization was needed because many linked libraries used MPI segments for the further built process. However, updating parallel communication channels was not the only necessary measure regarding the Magpar rebuild. Over time, the Linux kernel received significant changes, which had devastating effects on the Magpar

installation process. In short, during 2017, Magpar became broken. It was discovered that system-specific libraries merged while others became obsolete. Therefore, the project priority had to be refocused from the renewal of original communication libraries to complete the package restoration. Finally, the mentioned Magpar overhaul was estimated as one of the critical steps in making cloud communication HPC-efficient.

5.4.2 CPU Load Balancing

A host process scheduler randomly distributes all active processes with unique PID to available cores in the server. In the case of OpenStack cloud and Linux, this is the role of a completely fair scheduler (CFS). CFS assigns vCPU processes (VMs) to available cores in one iteration cycle and could re-assign them to other cores a few seconds later. This alteration is the result of load balancing algorithms, described in chapter 3.7.1.1. During the re-assignment procedure, CFS does not consider if two vCPUs of the same VM are located on the same physical CPU and could communicate via shared cache. Also, a sudden de-scheduling could occur if a fairness algorithm is triggered or system interruption is received. Consequently, frequent CFS re-assignment of vCPUs to different physical CPUs could lead to guest OS execution inefficiency.

Additionally, the switch of vCPU execution to another core requires a cache flush in the source core and re-establishment of frequently used cache lines in the target core. Each of these operations adds additional time that should be minimized as much as possible.

In order to solve the mentioned issues, we have used a technique called vCPU-pinning. This tuning procedure aimed to pin each vCPU to a unique physical core, limiting the CFS influence on VM performance. For this purpose, we have used the Linux “taskset” command, although other methods were possible as well. However, locating the exact PID of each vCPU was quite a challenge since vCPUs are light-weight processes, and different versions of libvirt use different file or folder structures.

Our initial measurements showed a ten percent performance improvement, but this was only in the case of two VMs. Of course, scaling up to a higher number of

parallel processes could only increase the scheduling efficiency and, consequently, cloud execution performance.

5.4.3 Inter-VM Intra-Server Communication Improvement

The inter-VM measurements in chapter 4.2.2 showed very low communication speedup and efficiency in guest OS solver execution (Table 2). Also, the rapid saturation with the number of processing elements was observed, which was not a good indicator of system scaling. Next, in chapter 5.2, it was revealed that one of the server's essential communication mediums – shared memory, remained unexploited. This solution, which was based on VM isolation, prevented VMs to "talk" directly to each other and forced data exchange through the complex set of cloud interfaces (figure 15). To solve this issue, we proposed the engagement of ivshmem, a PCI-based solution that directly maps guest OS Registers to the host's main memory. Since all VMs activate the same POSIX driver on the host, this mechanism allows that all guest instances can access the same host's shared memory region. As a result, this communication upgrade allowed efficient inter-VM communication on the same server by cutting excessive software overheads. This simple solution enabled direct communication of multiple MPI processes via shared memory, which is the fastest IPC.

However, the introduction of ivshmem occurred in several stages. First, it was necessary to activate the ivshmem feature by manually modifying the XML configuration file of a VM. This procedure turned out to be a significant challenge since the default emulator (QEMU) did not support the ivshmem device, while libvirt libraries lacked proper QEMU synchronization. The solution was to manually install newer QEMU and libvirt versions that effectively cooperate and then edit each VM configuration file to accommodate these changes.

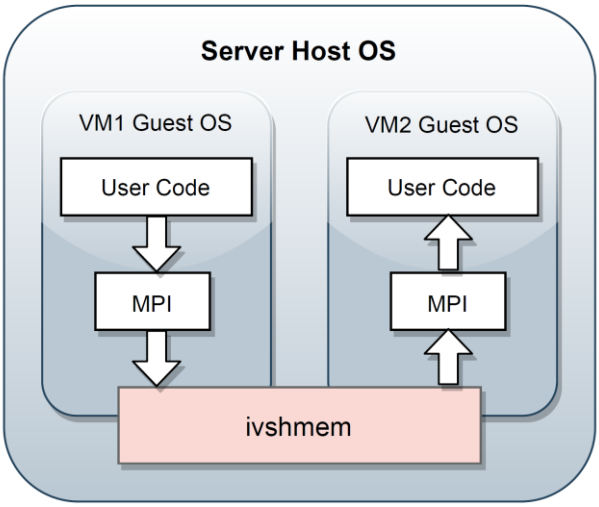

Figure 17: Proposed solution for inter-VM communication via MPI and ivshmem.

Second, the proof of concept was needed to demonstrate the shared memory communication advantage. However, similar to Magpar libraries, the original shared-memory synchronization mechanism was broken, and proper restoration was needed. This mechanism, called shared-memory server (SMS), ended up inoperable due to regular Linux updates, and its source code required patching. Therefore, to quantify the impact of ivshmem on inter-VM communication, we have developed several HPC performance tools and benchmark tests. Based on message size, our results showed execution time improvement up to the factor of ten, which was significant.

Third, once the ivshmem communication tests were completed, the solution merging MPI libraries with native ivshmem deployment was needed. It was discovered that with certain modifications, software called MVAPICH2 could be used for rebuilding the Magpar. The implementation details and rebuild nuances are given in chapter 6.4.2. Finally, repeated Magpar measurements demonstrated substantial performance gains compared with the initial test shown in chapter 4.2.2.

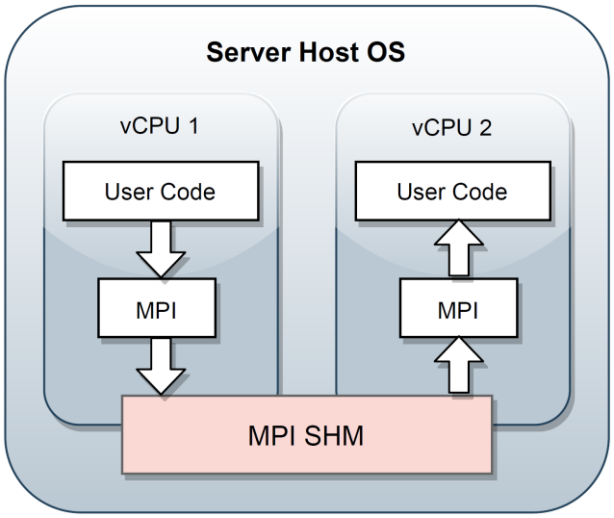

Figure 18: Proposed solution for inter-vCPU communication via MPI and ivshmem.

On the other hand, the prospects of Inter-vCPU communication improvement will not be further discussed. Our repeated measurements for all solvers in different hardware setups showed a little distinction between intra-VM communication channels. Figure 18 depicts the standard MPI shm channel without ivshmem that is automatically engaged by MPI. In this case, it was impossible to distinguish between ivshmem and TPC/IP sock channel, at least not beyond the measurement error. The reason for this equivalent performance lies not only in the activation of the shared cache memory of the host CPU but also in the efficient hardware acceleration – VT-x. These two hardware components allowed near-native execution speeds, which is a great result but only in a single VM cloud scenario.

5.4.4 Inter-Server Communication Improvement

5.4.4.1 Proper Infiniband Connection

One of the first and logical steps in inter-server communication improvement was replacing old 1Gbps Ethernet cables with the modern Infiniband connection. The host channel adapters (HCA) used are Mellanox MHQH19B-XTR cards, coupled with corresponding QSFP copper cables. The switch of the same manufacturer, Infiniscale IS5023, is selected for setting up this affordable and yet high-performing configuration. This time, a novel OpenStack software interface called VXLAN [88]

was installed to exploit this Infiniband hardware setup maximally. Although cloud data transfer occurred still via IPoIB, initial measurements showed a significant boost compared to all previous installations. As a result, even one-fourth of the theoretical maximum, which was 40Gbps for this HCA, proved to be a considerable upgrade to our cloud network.

The other aspect of this implementation, as mentioned earlier, was a cost reduction. It is estimated that engaging the latest chipsets that support SR-IOV is still too expensive for broad research groups, although it allows direct, remote memory access (RDMA), at least according to specification. Finally, the legacy Ethernet interconnection was modified into a service and maintenance network, allowing further cloud improvement for efficient inter-VM data exchange.

5.4.4.2 MPI process Distribution Planning

Since inter-server communication engages all 3-level hierarchies of data exchange, appropriate process scheduling becomes increasingly important. Namely, MPI does not differentiate between processes because they are all equally weighted during application initialization. On the other hand, the micromagnetic system segmentation makes a clear distinction between adjacent and remote partitions. As a consequence, the proximity of mesh segments directly determines the intensity of IPC during solver execution. Although this effect is not significant when inter-VM communication occurs within a single compute node, it gains momentum when inter-server data exchange is engaged. For instance, if two MPI processes that correspond to adjacent mesh partitions are distributed on the remote servers, their intensive communication will lead to colossal performance drops.

Generally, the effects of the described scenario could not be avoided entirely, particularly with small numbers of processing elements. Therefore, the importance of proper MPI process scheduling and distribution also grows with the increase of parallel computing elements, such as vCPUs. Since MPI cannot measure the quality of each communication channel and assign processes adequately, the maximum inter-server execution efficacy will never be reached by default mpirun.

Fortunately, the documentation study revealed that the MPI process distribution could be precisely controlled if a unique, user-created host file is used. This file could be later called along with other simulation parameters. Finally, our tests showed performance drops of over 350 percent if random process scheduling is manually enforced.

5.4.4.3 Load Balancing and CPU Isolation

Depending on the simulation load or hardware configuration, the proposed solution for sudden process de-scheduling, vCPU-pinning, does not always yield promising results. This rare scenario could be sometimes noticed in Controller nodes, which are running a plethora of services, software agents, daemons, and processes in general. Since all these services are distinct processes with low CPU times, fixing vCPUs to specific cores could lead to non-optimized scheduling. By design, these processes need to be active permanently while the process scheduler treats each one of them as equally weighted. In an HPC setup, this could be a problem as a vast number of active processes disrupt efficient CPU time calculations due to the CFS fairness algorithm. Consequently, our measurements with the new configuration revealed that vCPU-pinning could even degrade the original cloud performance, which was unacceptable.

In order to solve the mentioned issues, a new approach called CPU Isolation is suggested. CPU Isolation is a boot loader setting that enables excluding selected CPU cores from the process scheduler reach. As a result, CFS could not see these cores after system startup, although they are still accessible to the user. This system reconfiguration was a critical step preceding vCPU-pinning, as the new setup allowed full user control and freedom in processor allocation. Also, by performing CPU Isolation, it was possible to leave only a single core (core0) for kernel activities, whiteout influencing the overall system stability. Since the remaining Controller node processes require less than one percent of total CPU time, the entire system could be executed on the core0 with no effort. Therefore, the resulting cloud setup allows the ultimate efficiency in process scheduling because it permanently excludes virtual machines from CFS access. Alone with this tuning method, we managed to improve solver execution performance by over twenty percent, compared with original measurements.

5.4.4.4 MPI Communication Channel Renewal

Similar to intra-server measurements, the importance of Magpar's parallelization libraries renewal was one of the critical factors for inter-server communication improvement. However, the lack of shared memory allowed different MPI implementations to have a specific impact on data exchange between VMs. The influence of each MPI distribution on inter-server communication will be discussed in detail in chapter 6.5.3.

6 IMPLEMENTATION

Abstract

In order to implement proposed solutions for efficient micromagnetic cloud simulation, a three-stage set of processes is presented. The first stage, named ivshmem deployment, starts with a discussion of QEMU, XML, and libvirt orchestration challenges followed by ivshmem activation obstacles for various Linux distributions. The analysis continues with the breakdown of shared-memory access mechanisms, emphasizing the difference between spinlock and spinlock-free SHM synchronization. The first stage concludes with custom benchmark tools development based on restored blocking-read driver functionality, which is essentially a spinlock-free synchronization.

Once the extraordinary advantage of ivshmem over standard TCP/IP communication channel is confirmed, the second stage addresses the OpenStack cloud implementation. Moreover, a high emphasis is put on cloud tuning techniques, such as efficient network emulation, NUMA aware scheduling, and CPU load balancing.

Finally, after successful proof-of-concept and ivshmem cloud integration, the last step for effective micromagnetic cloud deployment could be actualized. As highlighted in previous chapters, the third set of processes include the restoration of a broken Magpar solver, upgrade of MPI communication libraries, Magpar-ivshmem integration, MPI process distribution planning, Infiniband hardware integration, and CPU load balancing optimization. Additionally, all processes are divided into two major sections: one for intra-server and the other for inter-server data exchange. This division was required because each setup contains nuances characteristic for its type of communication.

6.1 Ivshmem Deployment

Ivshmem implementation plays a crucial role in efficient cloud simulation execution, and therefore, a detailed deployment and activation of this shared memory mechanism will be given. As mentioned in chapter 3, ivshmem is a feature added in the QEMU source code, allowing direct guest access to the host's shared memory. It is estimated that by eliminating the VM isolation, a guest OS user could enhance the IPC by engaging a zero-copy data exchange mechanism. Ivshmem is implemented as an additional PCI device that could be activated before or after the VM creation. Since QEMU emulates all virtual hardware components while having host system privileges, it enables direct access from the ivshmem PCI device to the host's selected memory region. This process succeeds in two stages (figure 19).

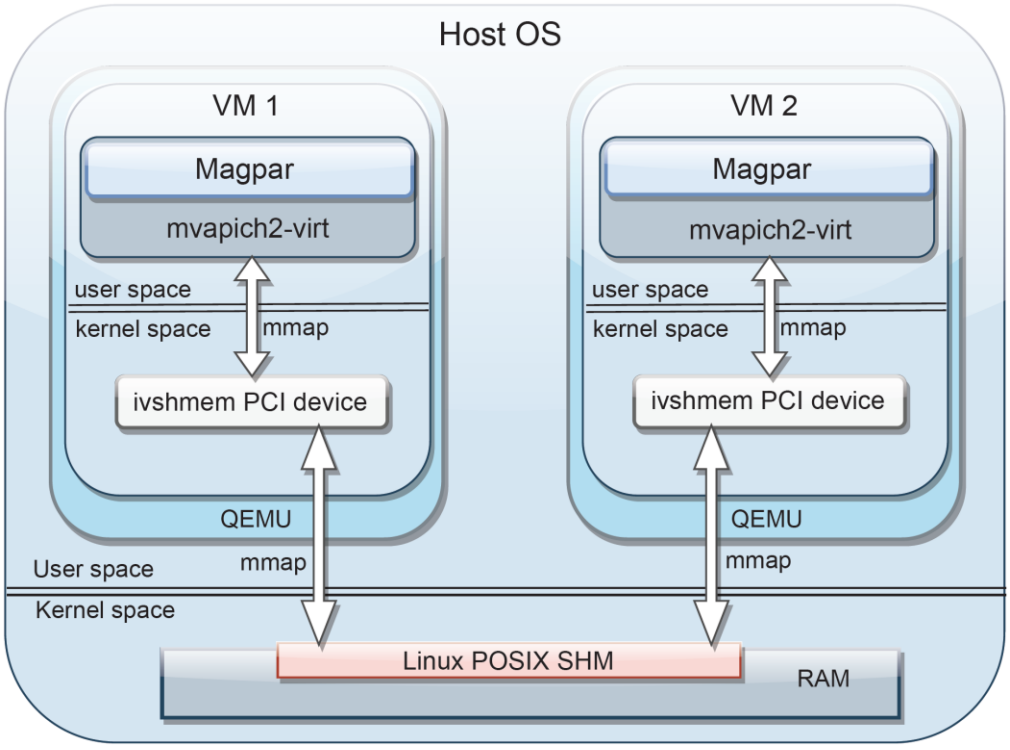

Figure 19: Ivshmem deployment in Linux.

First, there is a mapping between user application in the guest OS and the ivshmem (PCI device) memory region. This initial mapping is performed inside of

a virtual machine. The second mapping is done between ivshmem and Linux POSIX shared memory (SHM) on the host, created by QEMU during VM initialization. In both cases, a standard system call mmap() was used. Once the double mapping process is accomplished, a user-space application in guest OS gains direct access to the newly created POSIX SHM region of the host's main memory (RAM). Since this host's SHM region remains constant until manual deletion, each newly created VM could be attached to it. The only connecting requirement would be to use the same name as the created POSIX SHM, which in our case was "test-ivshmem". The process of adding the ivshmem PIC device to VM was designed to be a straightforward action. A user would manually create a POSIX SHM region in the host OS with standard Linux commands and then edit a VM configuration file by adding the following code to the <devices> section:

```
<devices>
  …
  <shmem name='test-ivshmem'>
    <model type='ivshmem-plain'/>
    <size unit='M'>256</size>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
  </shmem>
</devices>
```

The first line, *shmem name*, refers to the newly created SHM region within the host's main memory. A user then has to select one of the three types of ivshmem, which is ivshmem-plain in this example. Next, the sufficient size of shared memory has to be determined, which would ensure unconstrained data exchange. The given unit size of 256 MB was estimated as more than enough, as it could support simultaneous access to several dozens of VMs. Finally, PCI parameters are needed to complete the configuration, although only the slot number matters because others are generated automatically. The number 7 is selected, as the default configuration in OpenStack has the first six slots reserved for other devices, but this could vary among different VM managers. Suppose everything is done correctly, and VM is successfully re-created from the modified XML configuration

file. In that case, the following line should be added to the *lspci* command output in guest OS:

00:07.0 RAM memory: Red Hat, Inc Inter-VM shared memory (rev 01)

However, this was initially only a theoretical method for enabling ivshmem in virtual machines. In practice, many factors hindered the adoption of this promising shared memory communication mechanism. First, there is a requirement that QEMU supports ivshmem, which was not the case, at least not in the early stages of the project. Namely, during 2016/2017, OpenStack included only an old QEMU version in official repositories, which did not support ivshmem at all. In order to solve this issue, a workaround had to be found, requiring manual QEMU installation.

However, the user installation would disable all system privileges that came with default repository configuration and prevent standardized VM creation, such as using the Horizon dashboard. In addition, many other XML parameters of VMs have to be changed as well in order to accommodate the custom QEMU engagement. Second, even with working ivshmem, the proper shared memory synchronization mechanism was missing. The described procedure only allowed the creation of SHM between VMs, but it could not ensure proper race condition handling and prevent data corruption. The solution to this problem will be described in the next chapter.

6.2 Ivshmem Access Synchronization

Although the development of QEMU and libvirt allowed the gradual deployment of ivshmem, it did not provide an efficient shred memory synchronization mechanism necessary for most applications. This scenario was particularly noticeable for HPC applications because they require the highest communication performance standards. Consequently, the only option available to users and software developers for engaging ivshmem was applying the so-called spin-locks, which are non-blocking synchronization mechanisms for SHM read/write access. The idea behind spin-locks was continuous polling of a program condition until SHM access is completed. The principle of this mechanism could be explained in the example below (figure 20).

```
1   /*Sending process A (rank = 0)*/
2   if(rank == 0){
3       if (__sync_bool_compare_and_swap(ptr->rcvB, 0, 1) ) {
4           t_start = MPI_Wtime();
5           /*rcvB=1*/
6           /* Process A is writting */
7           for (i = 0; i < MAX_SIZE; i++) {
8               ptr->winB[i]=2;
9           }
10      /*Proces A (rank=0) has finished writting */
11       __sync_add_and_fetch(ptr->rcvB, 1);
12          while (! __sync_bool_compare_and_swap(ptr->rcvB, 0, 0)){
13              pthread_yield();
14          }
15      t_end = MPI_Wtime();
16      }
17      latency = (t_end - t_start)*1.0e6/2
18  }
19  /*Receiving process B (rank = 1)*/
20  if(rank == 1){
21      while (! __sync_bool_compare_and_swap(ptr->rcvB, 2, 1)){
22          pthread_yield();
23      }
24      /*rcvB=1*/
25      /* Process B is reading */
26          for (i = 0; i < MAX_SIZE; i++) {
27              priv_buf[i]=ptr->winB[i];
28          }
29      /*Process B has finished reading */
30      __sync_sub_and_fetch(ptr->rcvB, 1);
31  }
```

Figure 20: Spin-lock ivshmem access synchronization mechanism.

Communication occurs between two processes, the sending process A and the receiving process B. Each process contains its receiving window (e.g., winB), and the control bit (e.g., rcvB), which denotes the current state of the shared memory. Besides the two main states of the control bit: “0” – shm access allowed (r/w not active) and “1” – shm access not allowed (r/w active), the third state, “2” is introduced to identify the end of writing and to inform the receiving process that reading should proceed. The data exchange starts with sending process A and the atomic operation __sync_bool_compare_and_swap (line 3) that checks if the control bit (rcvB) of the receive winB has the value “0”, and if true, replaces (swap) it with the value “1”. This marks the start of the timer event t_start = MPI_Wtime(),

and writing could begin (for-loop). Once the writing is completed, another atomic function __sync_add_and_fetch(ptr->rcvB, 1) (line 11), increments the value of rcvB to “2”, which is an indicator for receiving process B to start reading. So far, the code execution was very efficient, but the following sequence describes the very reason why the spinlock mechanism is unacceptable for HPC. Namely, once the writing is completed, process A further execution is hindered with continuous polling of the rcvB until process B read is completed (while-loop). Also, during this time, the corresponding CPU (core) is busy with repeated execution of the while-loop, which is a waste of computing resources. The same applies to receive process B, which could not read from the receiving window WinB until rcvB is in a proper reading state, i.e., rcvB=2. The process B finishes the reading by decrementing the active-reading value of control bit (rcvB=1) to “0”, with another atomic function __sync_sub_and_fetch(ptr->rcvB, 1) (line 30). As a result, a spinlock poling in process A: while (! __sync_bool_compare_and_swap(ptr->rcvB, 0, 0)) (line 12) could be finally released, end timer set: t_end = MPI_Wtime() and the latency calculated.

As mentioned earlier, the downside of this approach is the complete reservation of CPU power for a poling completion event. Instead, the CPU should be engaged in executing other tasks, which was not the case. Because of this inefficiency expressed in the unnecessary consummation of CPU cycles, spin-lock synchronization served more as a demonstration rather than a practical complement of ivshmem. In order to solve the performance issue, a new synchronization system called shared-memory-server (SMS) was developed and offered as a third-party application.

The role of SMS was the creation, enumeration, and handling of special PCI registers of ivshmem that were created by QEMU. Also, SMS served as the central hub for inter-VM data exchange while replacing manual POSIX SHM creation. The resulting setup allowed sending synchronized interrupts from one VM to another by engaging the novel Linux IPC called eventfd. This feature made a huge difference as signaling between virtual machines was not possible initially due to VM isolation. By enabling interruptions, SMS allowed a blocking read feature of device files, which could be used for SHM access synchronization. This time, instead of endless “spinning”, the running process could wait for an interruption in the blocking read phase. Once the interruption is received, the waiting process is

unblocked, which signals the end of SHM access. Moreover, during the blocking phase, the process scheduler could assign another process to the logical CPU unit (core) instead of endless poling for the end-of-sync event. As a result, this would free computing hardware for other essential tasks.

However, establishing spin-lock-free synchronization required the application of separate device divers and modules called User-IO (UIO), which are not supported with every Linux distribution. In addition, UIO components are poorly documented and could not be found in one software repository. This fact was discouraging since the whole SMS access synchronization concept was based on the blocking-read functionality of the UIO device file. Consequently, an incomplete UIO setup resulted only in process execution blocking, while receiving interruption had no effect, which was unacceptable. Additionally, it was the user's task to write and integrate a user-part of the UIO driver into his application, which could only work if the correct kernel UIO module is loaded first.

Furthermore, existing MPI applications have to be rewritten because MPI does not support blocking read mechanisms of UIO drivers. Finally, regular Linux development introduced important changes into system register definition, and the unblocking feature of UIO became permanently disabled. Consequently, the whole spin-lock-free synchronization concept became useless as even a standard read() function performed on the UIO driver file could not block the code execution.

Although some solutions proposed by different authors existed, including a patch suggestion from the author of ivshmem, the overall situation was still difficult. Namely, QEMU and libvirt development succeeded independently, and many versions could not mutually cooperate. This incompatibility was particularly noticeable with the addition of extra features to VMs, such as ivshmem. Later, the overall situation improved when SMS came into the jurisdiction of the QEMU development team. SMS changed the name into ivshmem-server (figure 21) but, unfortunately, remained in the experimental phase.

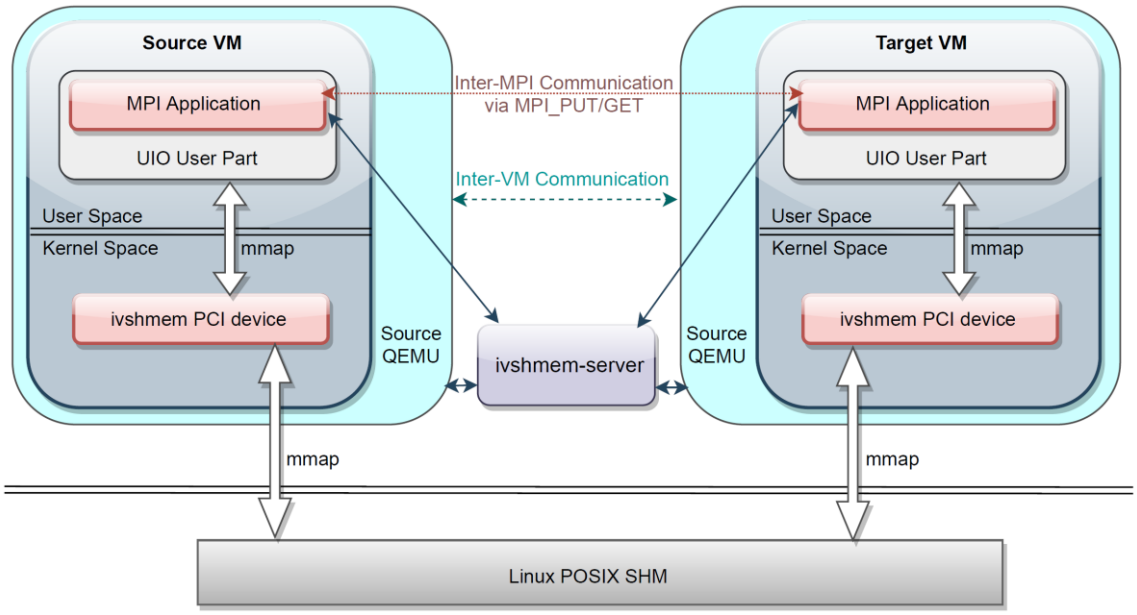

Figure 21: Architecture of ivshmem with an ivshmem-server synchronization.

To summarise, we patched and successfully loaded the UIO module, together with its ivshmem kernel counterpart, and solved the broken libraries issue. Also, proper versions of QEMU and libvirt are found that cooperate with each other in the latest stable version of Linux Debian architecture. Furthermore, we have created a custom performance test-code, integrating ivshmem, UIO user drivers, and MPICH parallel libraries to demonstrate the efficiency of ivshmem-based communication.

6.2.1 Ivshmem Access Synchronization Mechanism

In order to use ivshmem together with its synchronization mechanism - ivshmem-server, two consecutive inter-process communication (IPC) mechanisms have to be engaged. The first IPC occurs during VM creation between ivshmem-server, which is a single process on a host, and the UIO-ivshmem driver in guest OS. Since this exchange happens only once, there was no requirement for the most efficient IPC form but rather the most available one. Therefore, the Unix Sockets are engaged as the majority of OS platforms support them. The downside of this approach was the required modification of the XML configuration file, which was inconsistent among different Linux versions. As a result, many variants of ivshmem

parameter settings appeared over time, with, unfortunately, poor documentation. The second and more critical IPC occurs between two QEMU processes, and it heavily relies on kernel mechanisms. Since Linux already has built-in process monitoring, this IPC becomes very efficient as it avoids constant event poling. Also, the fact that QEMU processes are in user-space simplifies the communication requirements and allows for a built-in Linux mechanism, such as eventfd.

The given inter-QEMU IPC could be described in the following way: as soon as the QEMU process is triggered by an eventfd from another QEMU, it raises an interrupt in its VM with the help of KVM. This interrupt then triggers the ivshmem PCI register response, which is at the core of unblocking the process execution. We have also employed message-signaled interrupts (MSI) because they have more signaling options than original and relatively simple pin-interruptions. Thus, the resulting setup provided a mechanism for sending system interruptions from one VM to another, which was not possible previously due to VM isolation.

However, the integration of ivshmem with MPI required a completely new build of communication channels that will be based on ivshmem-server synchronization. This novel MPI-based channel would be equivalent to the MPI's standard single-side communication functions, such as MPI_Put() or MPI_Get(), which is important for the following efficiency comparison. Finding an appropriate synchronization mechanism among existing MPI options was also essential to precisely estimate our ivshmem-based communication channel. It is found that MPI_Lock() has the most similar features with our blocking-read implementation, although other sync mechanisms such as MPI_Barrier() or MPI_ WIN_FENCE() could be used as well.

Before making any tests, patching user and kernel parts of the UIO driver was necessary because of two significant tasks it accomplishes: a) It maps ivshmem PCI registers into the address space of the user application so that the modified MPI can access them. b) It sends and receives interrupts between VMs, which in turn call the MPI application processes. In the case of message sending, the blocking read() required for ivshmem access synchronization is performed in VM on a device file at the target.

As explained earlier, PCI-device (ivshmem) register mapping into guest-OS user-space is achieved by mmap(), which ensures direct shared-memory access from the user code. In an effort to simplify the implementation and API adoption, the

ivshmem PCI device was developed following standard PCI driver requirements and register setup. However, only two central registers are important for IPC and interruptions sending:

1) PCI-BAR0 or base-address register, pointing to a set of registers, including the so-called doorbell register.
2) PCI BAR2 register, pointing to the shared memory (SHM).

As a result, unblocking a target HPC application is accomplished by writing in our benchmark code an appropriate value to the ivshmem doorbell register, accessible by the application and MPI. This change in the doorbell register is automatically detected by the underlying QEMU process, which triggers inter-QEMU IPC and sends interruption to a target process. The principle of this synchronized data exchange mechanism could be well described with the example below.

Similar to the spin-lock data transfer example, the communication based on UIO blocking read functionality occurs between the sending A and the receiving B process. Additionally, sending includes ten transmissions tracked by the counter (line 3), which will be later averaged during latency calculation (line 16). Once the writing process is completed (lines 6-8), process A is sending an interrupt signal to the blocked (line 22) B process by writing into the Doorbell register (lines 9,10). This is a key difference to the spin-lock mechanism because the blocking read function (line 22) blocks the execution of process B until interruption is received. Compared to spin-lock, “blocking” is much more efficient since the process scheduler could assign its CPU (core) to another process and therefore continue the program execution. After sending the interruption, process A blocks itself (line 11) while waiting for the reading completion on the other side. This allows feedback information about data transmission success, similar to other shared-memory syncing functions, such as MPI_Lock(). Once the write-read-feedback session is completed, sending is repeated until the counter reaches its exit condition (line 3), and the end-time of transmission is recorded (line 15). It is also worth noting that the measured latency is divided by two because it includes the message reading confirmation.

```
1   /*First sending process A (rank = 0)*/
2   if(rank == 0){
3       while (memptr->cnt < 10) {
4           if (memptr->cnt == 0) t_start = MPI_Wtime();
5       /* Process A is writting */
6           for (i = 0; i < MAX_SIZE; i++) {
7           memptr->winB[i]=2;
8           }
9       msg = ((A_dest & 0xffff) << 16) + (cmd & 0xffff);
10      map_array[Doorbell/sizeof(int)] = msg;
11      rv = read(fd, &buf, sizeof(buf));
12      /* Atomic operation on counter */
13    __sync_add_and_fetch(memptr->cnt, 1);
14      }
15  t_end = MPI_Wtime();
16  latency = (t_end - t_start)/(10*2);
17  bandwidth = (MAX_SIZE*32)/(1.0e6*latency);
18  }
19  /*First receiving process B (rank = 1)*/
20  if(rank == 1){
21      while (memptr->cnt < 10) {
22      rv = read(fd, &buf, sizeof(buf));
23      /* Process B is reading */
24          for (i = 0; i < MAX_SIZE; i++) {
25          priv_buf[i]= memptr->winB[i];
26          }
27      /*Process B has finished reading */
28      msg = ((B_dest & 0xffff) << 16) + (cmd & 0xffff);
29      map_array[Doorbell/sizeof(int)] = msg;
30      }
31  }
```

Figure 22: Blocking-read ivshmem access synchronization mechanism.

As a result, a new SHM communication channel based on single-side MPI_Put memory access is created that could further replace slow and inefficient TCP/IP inter-VM communication. Finally, we have achieved that blocking read implementation of ivshmem works for the first time with QEMU version 2.9.50, MPICH version 3.2, libvirt version 2.0, and Ubuntu version 16.04.

6.3 Ivshmem and Cloud Tuning

Although OpenStack instances are created by KVM and QEMU as every other virtual machine, activating ivshmem in this cloud OS is far from straightforward. As mentioned earlier, the QEMU development surpasses the Linux release or update schedules, and obtaining the latest emulator features is usually a long waiting process. Besides QEMU, another virtualization API – libvirt, also has an independent release cycle, which is unfortunate. Both of these software packages need to be synchronized in order to produce error-resistant virtualization. As a result, versions of QEMU and libvirt in official Linux repositories are sometimes years behind the current development state. Therefore, integrating the mature ivshmem release with OpenStack required the manual installation of QEMU from a third-party developer community. Since this installation was user-based, many system privileges accompanying the official repository update were lost. In general, this should not be a huge issue, but OpenStack is an open-source project used by many big companies, such as IBM, Dell, and HP, and security plays a vital role in its development. In order to customize this massive project and implement non-standard features such as ivshmem, many layers of user protection have to be individually modified. The main reason for this is that OpenStack runs on kernel privileges typically not available to ordinary users. For instance, besides standard user/admin access rights, additional Linux file-protection mechanisms, such as AppArmor [74] and Selinux [73], have to be bypassed to customize instance creation. Also, this procedure requires simultaneous system change of both QEMU and libvirt configuration scripts.

Unfortunately, setting up an appropriate combination of security settings and configuration adjustments was quite challenging due to insufficient documentation and substantial differences among Linux distributions. On the other side, engaging an ivshmem PCI device is not enough for HPC. Other factors can significantly influence the performance of code execution in both initial and computing/communication phases. These factors will be presented in the following chapter.

6.3.1 NUMA Aware Scheduling

Non-Uniform Memory Access or NUMA reflects characteristics of modern chipsets and presents the latest achievements in memory management. As the current hardware development trend shifted the focus from high clock speeds to an increasing number of CPUs and cores, it postponed approaching the physical limitations of contemporary silicon technologies. In order to address the resulting increase of memory load due to the high number of CPUs, a new solution dividing the main memory into several segments was found. Therefore, each segment or node remained within the main memory, but the access speeds varied based on the CPU proximity or local hardware arrangement. In short, each NUMA node received its local and remote set of CPUs, along with higher and lower access speeds, respectively.

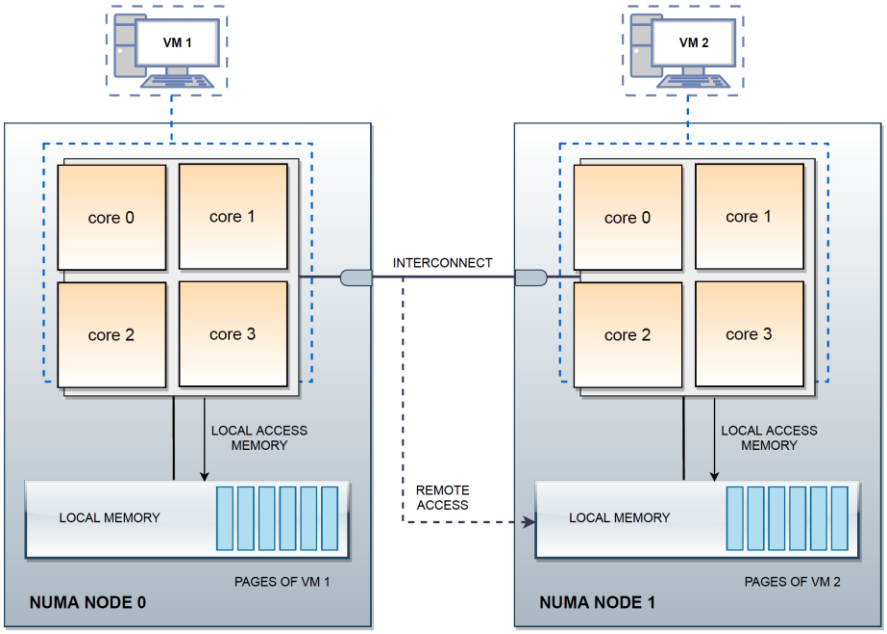

Figure 23: Example of 2 node NUMA deployment.

In virtualization, and particularly with multicore cloud servers, NUMA awareness became increasingly important as the memory access speeds differed by orders of magnitude among different VMs. The exact segmentation, i.e., precise core distribution, became the critical parameter required for differentiating local from remote memory access. With this in mind, the OpenStack instances could be

manually created via CLI and distributed to the appropriate NUMA node. Moreover, this solution is more time-efficient than the alternative method that implies binding each vCPU to a particular NUMA core individually.

6.3.2 Proper Network Emulation Selection

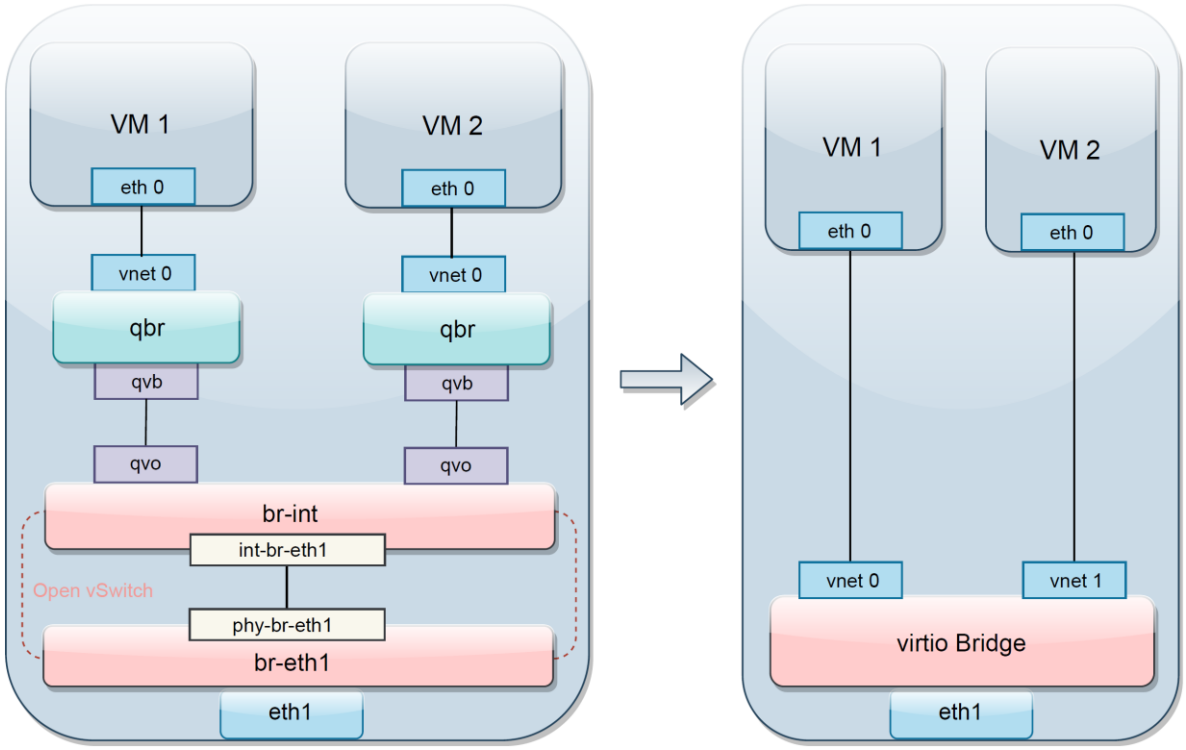

Figure 24: Virtio Bridge implementation instead of OVS.

The selection of appropriate network emulation plays a crucial role in virtualization performance, even when shared memory is used as the primary communication channel. The reason for this lies in the design of MPI initiation and synchronization procedures that are limited to standard TCP/IP protocol. As a result, a significant amount of data succeeds through the Neutron part of OpenStack (left side of figure 24), which amounts to the overall overhead described in chapter 5.2. In order to address this issue, a new approach was taken, which includes a more efficient IO virtualization method called virtio [64]. As described in chapter 3.5.2.1, virtio provides for paravirtualization, meaning that its device-driver virtio-net is aware that it is executed in VM. By engaging virtio-net, KVM avoids interception of device-

driver access to emulated PC devices. Instead, virtio call parameters are directly forward to QEMU.

The resulting virtual network configuration is presented on the right side of figure 24. The custom virtio Bridge (figure 24, left) was created to replace the complex system of Open vSwitch Bridges and interfaces that are causing unnecessary communication overhead. The management of virtual Bridges could be performed using the standard Linux brctl command, allowing not only Bridge creation, configuration, and deletion but also interface setup and attachment. The second, perhaps more convenient option for inexperienced Linux users is the engagement of libvirt (virtio) Bridge, which is automatically created upon activation of libvirtd daemon. Once the virtio Bridge is created and properly configured, the last step in cloud network-tuning would be the manual modification of OpenStack's XML file. This process starts with replacing OVS interfaces with newly created virtio counterparts and ends with a reboot of networking services. The effects of this change will be discussed in greater detail in chapter 7.2.

6.3.3 CPU Load Balancing

The distribution of CPU load is one of the key parameters determining the efficiency of an HPC system. Therefore, keeping the load balance between different cores and minimizing frequent scheduling/rescheduling of processes are critical requirements for achieving good simulation performance. With parallelization based on MPI libraries, built-in optimization prevents the host scheduler from a short-term, random distribution of MPI processes, which is excellent for applications running directly in host OS. However, an additional layer of virtualization present in the cloud eradicates the distinction between HPC processes and standard Linux processes, directly influencing the system load balance. Namely, vCPU processes corresponding to each VM core are visible to the Process Scheduler (CFS) as a child or light-weight processes of the primary (parent) VM process. As such, they are prone to sudden de-scheduling that could place the vCPU host process back into the waiting queue. Moreover, this could result in constant rescheduling of vCPU host processes to different cores or logical processing units, which is terrible for performance.

In order to overcome this characteristic of CFS, the method known as vCPU-pinning was applied. It consists of the bounding of each host's vCPU process to a particular core, preventing the scheduler from accessing and putting them in the scheduling queue. However, locating vCPUs is not an easy task as they are not visible among standard Linux processes due to their "light-weight" nature. Also, their exact process ID (PID) could only be found in OpenStack's temporary files, and it is highly dependable on particular Linux distribution. An example for CentOS 7, vCPU search (grep) output is given below:

```
sudo grep pid /var/run/libvirt/qemu/instance-00000033.xml
<domstatus state='running' reason='booted' pid='1977'>
    <vcpu id='0' pid='1984'/>
    <vcpu id='1' pid='1985'/>
    <vcpu id='2' pid='1986'/>
    <vcpu id='3' pid='1987'/>
```

Once the correct PIDs are obtained, the next step is CPU-pining. For this project, the Linux *taskset* command was used, but other methods are possible as well. Finally, this manual setup could be time-consuming in multiple VM scenarios, and a simple bash script that could automate this process is highly recommended.

6.3.4 Summary

The described tuning methods, together with the renewal of SMS-based shared-memory access synchronization, allowed us to run, for the first time, ivshmem in the OpenStack cloud. Our bandwidth and latency measurements showed that it is possible to achieve performance improvements of three to six times compared to standard TCP/IP. In the next chapter, the final step of integration between simulation software (Magpar) and ivshmem shared-memory will be given.

6.4 Magpar Enhancement

In order to efficiently integrate Magpar with ivshmem and run it on a private OpenStack cloud, previous network-tuning processes are combined with new optimization techniques and measures. As mentioned earlier, developer support for Magpar stopped in 2010, and after recent Linux updates, the original version with obsolete mpich2 libraries became broken. Therefore, our first step was to restore the solver and repeat measurements to reestablish the correct simulation outputs in the updated cloud. In an effort to accomplish this, all Magpar installation scripts had to be recreated, including fixing broken links, finding proper library replacements, and reorganizing folder structures. Finally, Magpar had to be reconfigured appropriately, rebuilt, and tested for compiler errors.

However, since Magpar was designed for early iterations of the MPI-2 standard and MPD process manager, the decision was made to improve the communication libraries to the highest version of the MPI-3 standard. At that moment, the latest stable version of the MPI-3 standard was mpich3.2.

In the final step, Magpar was merged with MVAPICH2-virt, which enabled shared-memory communication between OpenStack cloud instances.

6.4.1 Magpar Integration with MPI-3 Standard

Upgrading Magpar to the latest stable version of MPI-3 standard - mpich3.2 was significant because it solves two main issues attributed to the original software package:

1) It removes the multi-purpose daemon (MPD) and its obsolete process manager with a more advanced successor called Hydra. Not only is Hydra newer and better optimized, but it also provides additional features, such as topology-aware process binding and an integrated process management interface. The latter means that the process manager (Hydra) is now integrated into the *mpiexec* script, and the previous configuration, creation, and maintenance of the MPD ring is no longer necessary. This change is important because it makes the running of MPI applications a straightforward process, which does not require an additional technical background.

2) It introduces a more efficient and optimized MPI communication channel called Nemesis, which integrates all previous solutions such as shared memory (shm) and TCP/IP socket channel (sock). As a result, our measurements showed clear improvements compared with early versions of the mpich2 package, particularly when the multi-process scenario is engaged. However, the shm Nemesis channel utilization is only possible within a single VM, and it could not be applied to multi-VM parallel execution, which is the essence of a cloud.

6.4.2 Magpar Ivshmem Integration

Although upgrading Magpar to the MPI-3 standard brings noticeable improvements in efficiency and software usability, achieved communication speeds were not suitable for HPC because inter-VM data exchange still relied on TCP/IP. Unfortunately, mpich3.2 was not designed with ivshmem support, and for this reason, it cannot be used for shared memory communication between cloud instances. Also, using our solution described in chapters 4.1 and 4.2 is not feasible because it would require reverse engineering of all Magpar parallel libraries and the creation of a new MPI communication/synchronization channel accordingly. Furthermore, because MPICH is a very complex software package with many interlinked libraries, functions, and calls, creating our custom, ivshmem-enabled MPI version would require time resources beyond this project's scope. Therefore, after extensive research, it is concluded that after certain modifications, we would be able to use a solution from Ohio State University called MVAPICH2-virt. Originally, MVAPICH2-virt was designed for SR-IOV [61] enabled hardware that we do not have, but on the other hand, it has built-in support for ivshmem, which we used before. SR-IOV is a hardware accelerator integrated today only in the latest server units and is thus not available to most research groups. It enables direct access of VM drivers to the physical hardware by avoiding slow QEMU/KVM emulation and provides highly efficient inter-server Infiniband communication. However, during intra-server communication, SR-IOV lags behind the ivshmem because of additional overhead created with new interfaces and software Bridges.

Because MVAPICH2-virt does not come with the source code and direct Ubuntu support, we were forced to engage additional Linux tools to convert RPM packages into Debian-based counterparts. Additionally, standard Linux OFET verb libraries

have to be deleted and replaced with special Mellanox verbs in a poorly documented process. Furthermore, beyond installation script modifications mentioned in chapter 4.3, the procedure of rebuilding Magpar with MVAPICH2-virt required a change in the MPI compiler script. This modification was needed for establishing correct paths between new MPI libraries and Ubuntu system binaries. Finally, running ivshmem-enabled Magpar could not be performed before creating the ivshmem guest PCI driver by modifying host XML files. Therefore, with proper parameter settings documented by us, we have achieved performance improvements between factors of 1.4-6, depending on the number of engaged vCPUs. In particular, we have accomplished that Magpar is usable again in its original form. Moreover, we improved it further with MPI-3 standard and enabled for the first time a highly-efficient ivshmem inter-VM execution.

6.5 Inter-Server Communication Improvement

6.5.1 Ethernet Replacement

The first and obvious step for inter-server communication enhancement was replacing old Ethernet cables and switches as the backbone of our cloud data network with Infiniband.

However, without corresponding hardware accelerators, such as SR-IOV, the full potential of the RDMA Infiniband feature could not be exploited before hitting throughput limitations of data encapsulation. Therefore, cloud Compute nodes lacking the latest chipsets or motherboards have to be connected through IPoIB, while the management network could still operate via Ethernet. Fortunately, Neutron's contemporary project development allows replacing older GRI tunneling with VXLAN, leading to performance upgrades. VXLAN ensures encapsulation and propagation of the Ethernet frames in the same form they have left the VMs. This data transfer occurs over the so-called overlay network, which is Infiniband, in our case.

Even with the slightly slower hardware configuration, the range of improvements made by introducing the VXLAN Infiniband overlay network starts with more than sixty percent before reaching the factor of eight. Moreover, when a higher number

of parallel processes is engaged, i.e., eight or more, Infiniband's advantage is unquestionable because large numbers of vCPUs lead to complete congestion and endless execution in the Ethernet environment.

The following chapters will show that with particular enhancements and tuning techniques, a speedup in inter-server communication could be achieved even with the relatively smaller simulation system size, which was previously unimaginable.

6.5.2 MPI Process Distribution Planning

In chapter 5.4.4, proper MPI process distribution is emphasized as a necessary simulation tuning, particularly relevant for inter-server execution. Also, it is stated that the influence of parallel process arrangement grows with the number of processing elements (vCPUs). In order to illustrate this fairly simple, yet very effective setup, the following host file (host-file) example will be used:

```
cat ./host-file
node1:4
node2:2
node3:2
```

As seen above, three distinctive VMs are used, each corresponding to a different node, and at least one of them is located on a remote server (e.g., node3). In addition, if the list of all nodes (VMs) located in /etc/hosts is not established, IP addresses could be used instead. In essence, this simple setup regulates two main parameters: 1) the order of the nodes, which corresponds to MPI process enumeration, and 2) the number of parallel MPI processes per node, determined by the value after the colon (node1:4). Since we established earlier that VM1 (node1) and VM2 (node2) are located on the same server, the following process distribution shall occur: MPI processes 0-3 (4 in total) are spawn on the node1, processes 4-5 on the node2 and processes 6-7 on the node3. By placing the remote (node3) server on the bottom of the list, we have minimized the inter-server communication, which is an essential step associated with the level-3 communication hierarchy. Finally, the precise setup of the MPI processes per node

prevents the random distribution of adjacent processes to other nodes, which was detected during our measurements. For instance, the MPI process enumerated with number 2, instead of on node1, could end up on node2 or even node3. Since the highest communication rate occurs between neighboring processes, this scenario would result in a considerable performance drop. Therefore, by following these two principles and engaging described setup, significant simulation efficiency gains could be achieved.

6.5.3 MPD Ring Replacement

The significance of replacing the obsolete MPD Process manager is already explained in chapter 6.4.1. It is shown that a new, integral solution called Nemesis has better usability and improved performance. However, these improvements could not be fully exploited unless the inter-server test scenario is engaged.

The improvements and optimizations made at the D3 (MPI) communication channel are evidently surpassing legacy implementation, particularly with higher numbers of processing elements. For instance, Magpar rebuilt with mpich3.2 could perform almost twenty-five percent better in the case of eight vCPUs than the MPD equivalent. This change is significant, and it is over five times better result than the best case of the intra-VM measurements, although the same communication medium is used in both cases.

It is also important to note that shared memory does not play any role in inter-server communication, and the benefits of engaging ivshmem could not be realized in this case. Therefore, the results and test cases demonstrated in this chapter are based on the latest version of MPICH, which was 3.2 at the time of measurements, although MVAPICH-virt performance with/without ivshmem followed quite closely.

6.5.4 Core Isolation and Inter-Server Communication

As shown earlier in chapter 6.3.3, process binding plays a significant role in the efficient execution of MPI-based applications. However, in standard virtualization models such as QEMU/KVM, the unmodified guest OS is running on top of a hypervisor, and user applications are unaware of the physical host configuration.

Therefore, each guest MPI process has to be scheduled two times, once in a VM, to an emulated core or vCPU, and the second time in a host OS, to a physical core or a thread. This means that the procedure of binding processes to cores or the so-called vCPU-pinning has to be performed two times in both OSs, which is tremendous work for setup with many parallel processes. Also, all of these actions have to be repeated after each system reboot, which only lowers the attractiveness of this performance tuning method. Fortunately, new process managers such as Hydra include extra features that automate CPU binding procedures, but this solves only a guest scheduling problem. The mentioned solution is called hwloc [89], a software package that provides a multiplatform abstraction of the hierarchical organization for underlying hardware architecture. In short, hwloc gathers all parameters such as shared cache, cores, memory nodes, GPUs, or network interfaces to exploit them most efficiently. Unfortunately, this solution can only work with the MPI spawn processes, which are not visible from the host OS. As a consequence, each host process binding has to be manually performed. Moreover, locating processes in a host OS corresponding to individual vCPUs is a complex task for several reasons:

First, these processes are not part of the standard Linux process list, but instead, they belong to the group of the light-weight processes that could not be directly listed with standard Linux tools, such as top or htop.

Second, the naming of these processes is not in direct correlation with their parent qemu-kvm process, so they could easily be mistaken with similar PIDs. Finally, the file location containing vCPU process information is hidden deep into the Linux system folder structure, and obtaining the data requires an experienced user. Fortunately, once the exact folder path is retrieved, a short script could perform vCPU-pinning tasks with no effort. The only input required from a user side is the full name of the VM that corresponds to the virtualization software, which is easily obtainable. For instance, OpenStack VMs appear in the host OS in the “instance-00000xx” format, where xx designates a particular VM.

Once the binding procedure is completed on both guest and host OS sides, each MPI process running in the cloud is then linked to a single physical core or thread and could not be de-scheduled. As presented in chapter 7.2.2.2, CPU-pining removes all the spikes or performance drops in measurements but cannot

substantially improve intra-host output results as other tuning methods. The best results with vCPU-pinning are achieved with powerful multicore machines, i.e., servers with sixteen or more logical processing units because Linux kernel and other system-critical processes have to be executed continuously. This scenario excludes competing for hardware resources, allowing process binding to generate significant performance gains, particularly in setups running many parallel nodes.

On the other hand, i.e., in the case of eight-core machines, vCPU-pinning could even lead to small drops of execution efficiency. This unwanted effect can occur in computing nodes running a vast number of services, agents, daemons, besides standard Linux processes, limiting the efficiency of CFS scheduling algorithms. Although vCPUs are pinned to a particular core and could not be scheduled elsewhere due to limited resources, CFS is forced to schedule also the other processes to these cores. Since pinned vCPUs could not be executed on adjacent cores, they have to be dropped from scheduling queues, causing them to miss a few waiting cycles.

Fortunately, a solution to this issue exists, and it could be achieved through CPU isolation. Among others, Linux OS enables isolation of individual cores during boot procedure, which would later be inaccessible to the scheduler during OS runtime. These “free” resources could be then assigned with special Linux commands, such as “taskset”, to desired applications or binned to VM processes in our case. Because of hardware accelerators such as VT-x, the afore-described procedure could enable practically native VM process execution speeds. Our results showed that it is possible to achieve up to twenty-five percent of execution speed improvement during inter-server measurements, which is substantial, considering that the data transport medium was not changed.

7 EVALUATION AND MEASUREMENT RESULTS

Abstract

Following the succession order of project development unveiled in chapter 6, the evaluated measurement results are divided into corresponding logical sections. The first section presents the bandwidth and latency measurements, compared between the ivshmem and MPI_Put channel, for both TCP/IP and shared memory settings. The following data analysis extends for a full range of message sizes, showing ivshmem superiority over standard communication channels.

The second section comprises the same measurement setup, only this time, numerous cloud-based optimizations and communication factors are included: network emulation, cache arrangement, CPU load balancing, OpenStack overlay, NUMA, and virtio activation. Again, ivshmem based improvements demonstrated clear superiority over any TCP/IP deployment.

In the third section, the original Magpar cloud deployment was restored and then compared with its enhanced version, comprising both ivshmem and the latest mpich3.2 communication channels. This test setup focused on solver efficiency evaluation during single host inter-VM measurements, where ivshmem confirmed its excellent cloud computing performance.

Finally, the last section addresses the inter-VM solver performance by closely examining inter-server communication characteristics. Besides Magpar restoration, this subchapter introduces physical medium changes from Ethernet to Infiniband, MPI process distribution planning, and a new approach to CPU load balancing. The achieved results enabled an increase of speedup and almost linear acceleration growth with parallelization, which was previously unimaginable.

7.1 MPI and ivshmem Channel Comparison

Performance measurements are performed in our most powerful server by varying block sizes and following three scenarios: a) an inter-VM communication with MPI_PUT for one-sided data exchange and MPI_WIN_LOCK for passive remote memory access (RMA). The setup engages the new MPICH Nemesis-sock channel and results in standard TCP/IP communication. b) An inter-VM communication with our MPI_PUT and MPI_WIN_LOCK equivalent via ivshmem and SHM synchronization. This setup required the integration of ivshmem in MPICH. c) A host OS communication with the standard MPICH over the SHM Nemesis channel. The third scenario was used as a reference for a) and b). Results for the elapsed time are depicted in figure 25, and for bandwidth, in figure 26.

The elapsed time is calculated at the sender by dividing by two the time difference between the first byte sent and the reception of a transfer-complete notification from the receiver, with SHM synchronization included. Bandwidth is calculated by dividing the elapsed time by the respective message sizes.

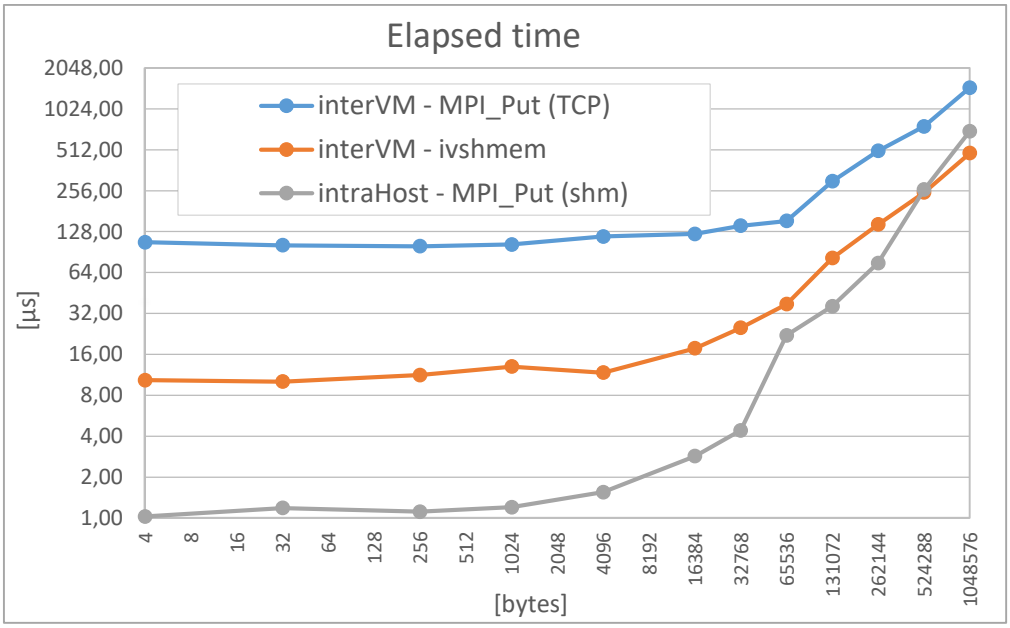

Figure 25: Measurements of elapsed time for various message sizes.

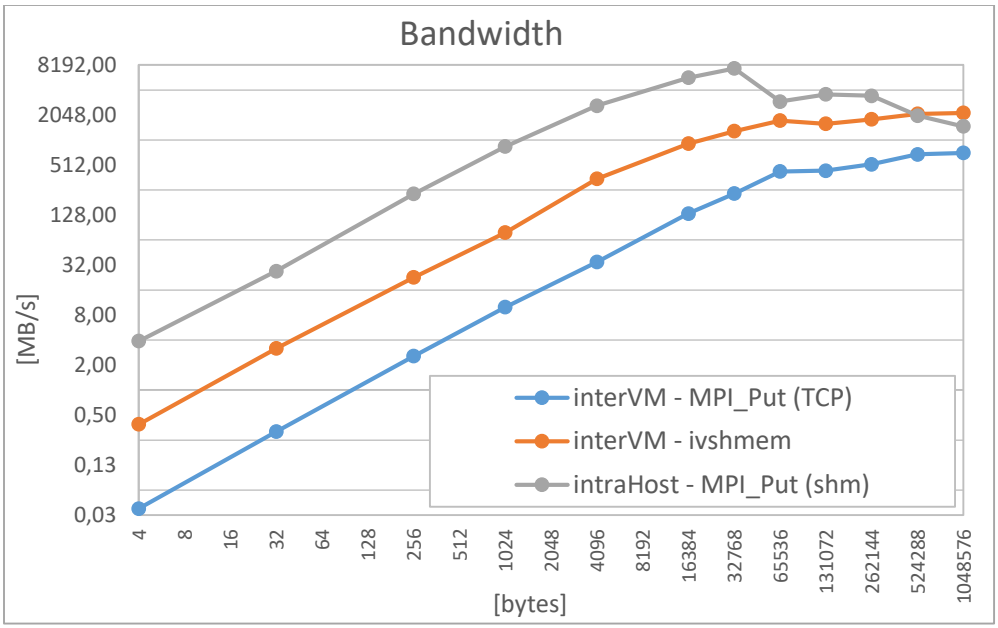

Figure 26: Results of calculated bandwidth for various message sizes.

Based on these measurements, we observe four phases in figure 25 and figure 26:

1) Phase 1, for block lengths of 0-4 kB per transferred message: in this phase, we can see that SHM synchronization dominates the data transfer time because the elapsed time does not increase with message size doubling. Furthermore, the amortization costs for SHM synchronization are declining with each increase in message size. During this phase, ivshmem surpassed TCP/IP by a factor of ten for both bandwidth and elapsed time.

2) Phase 2, for block lengths of 4-64 kB: in this phase, the elapsed time increases correspondingly with the message size, which can be observed on the ivshmem curve (figure 25). The maximum bandwidth of approximately 7.4 GB/s is reached at a block length of 32 KB, as well as a maximum value for TCP/IP linear slope (figure 26). The reason for this behavior is that only one TCP/IP packet is needed for transferring all message sizes. Finally, the elapsed time curve sharply changes its slope angle (figure 25).

3) Phase 3, for block lengths of 64-256 kB: in this phase, it could be seen that the transmission saturation is reached at a block length of about 256 KB, with ivshmem

a data rate of approximately 2 GB/s. Furthermore, ivshmem surpasses TCP/IP (blue line) by a factor of four.

4) Phase 4, for block lengths higher than 512kB: we can observe a declining reference curve, while ivshmem becomes the fastest communication. This development is a remarkable result because virtual communication can even be faster than real data exchange for huge block sizes. The reason for that surprising behavior is that ivshmem does not make any time-consuming system calls, contrary to host applications coupled by SHM. Also, ivshmem operates entirely in user-space because of the mmap and blocking-read feature, which is in the user space along with UIO. Because of its efficiency, ivshmem is second to none for HPC.

The performance difference between ivshmem and TCP in this phase is a factor of three, although TCP implementations show increased efficiency for large message sizes. Finally, it should be mentioned that the selection of synchronization/communication channel pair could shift a peak bandwidth to a different block length. For instance, with point-to-point MPI_Send and MPI Start-Post-Wait-Complete, the maximum TCP/IP bandwidth is already obtained at a message size of 256kB, while ivshmem reached four times higher throughput.

7.2 Cloud Tuning and ivshmem Integration

7.2.1 Tuning Standard TCP/IP Inter-VM Communication

For TCP/IP tuning in subsequent chapters, we used OpenStack Juno, Ubuntu 16.04 as guest OS, CentOS 7.1 as host OS, QEMU 2.9.50, libvirt 2.0, mpich3.2, and virtio 1.1.1 as software environment. Additionally, we data is sent from one MPI process to the other while varying its size from 4B to 1 MB. The transmission is accomplished using the MPI_PUT call over the standard (MPICH) Nemesis-sock channel. However, because host shared memory could not be engaged due to VM-isolation, MPICH emulates this functionality via TCP/IP by sending back and forth data packets carrying shared variables.

In the first step, the influence of level-3 caching on OVS is discussed compared to L2 caching only. As a reference, we used the elapsed time for transferring data in a configuration where no cloud OS was engaged (VM only). The results are shown in the light blue, grey, and yellow curves of figures 27 and 28. The two light blue curves depict the elapsed time and bandwidth for inter-VM data exchange with QEMU emulation of a fully virtualized network and no cloud OS. In this case, each guest device-driver access to the QEMU-emulated network device is intercepted by KVM, which is very inefficient. As one can see, if OpenStack is deployed with its paravirtualized drivers and OVS (grey curves), performance is much better than without cloud. The reason for this significant improvement lies in the concept of paravirtualization, where KVM no longer intercepts gest’s access to the emulated hardware. Instead, all parameters of gest device-driver calls are forwarded directly to QEMU. Additionally, engaging level-3 caching further improves performance (yellow curves) because of higher cache capacity and reduced access time. Similarly, replacing the server with a desktop PC (dark blue curves) also showed good performance, but the disadvantage was limited scalability and cloud incompatibility. In the end, the only measure that made a significant difference was the replacement of Neutron’s OVS by virtio-net.

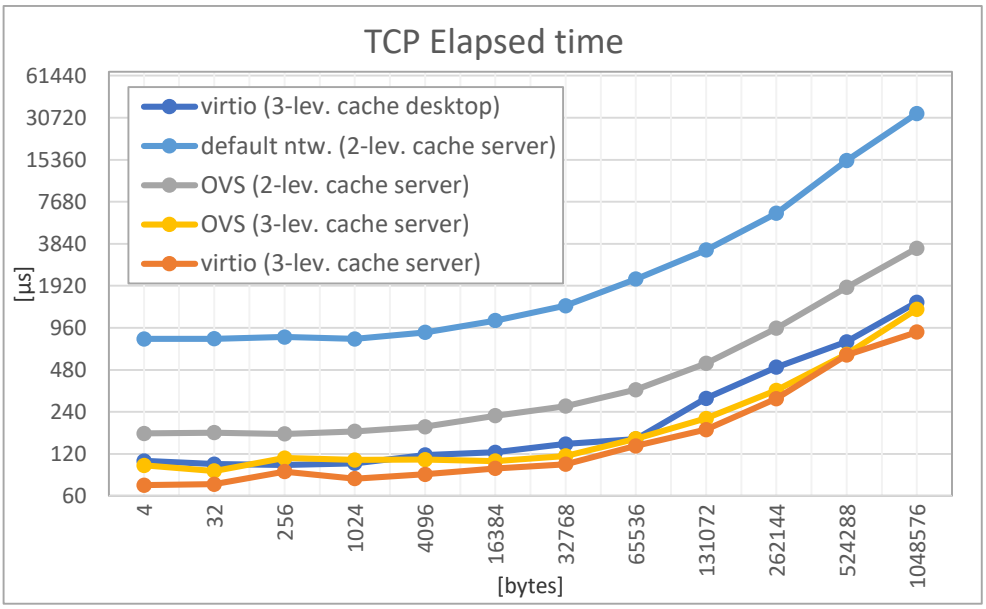

Figure 27: Elapsed times for TCP/IP-based inter-VM communication.

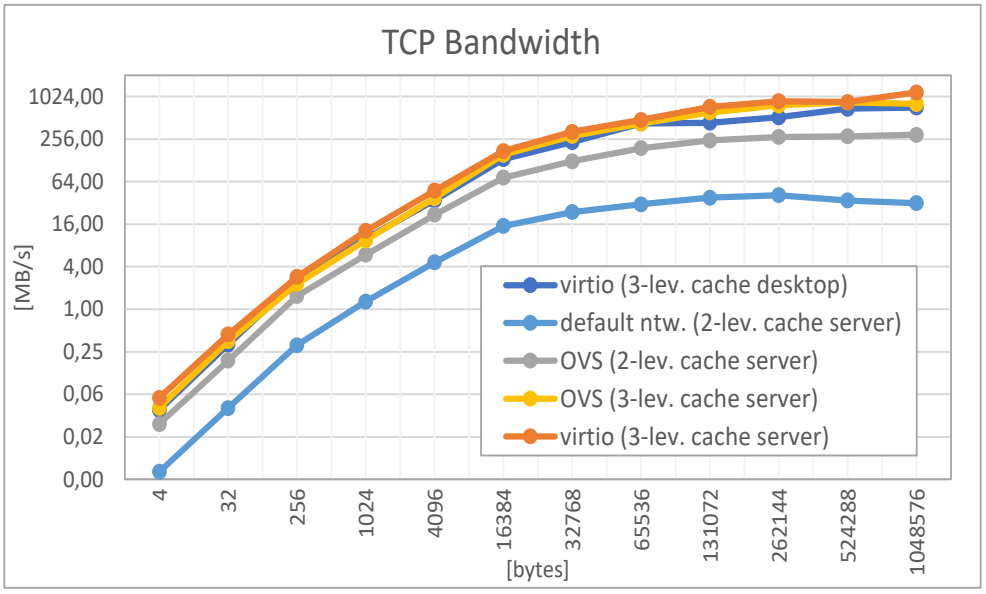

Figure 28: Bandwidth for TCP/IP-based inter-VM communication.

7.2.2 Tuning the ivshmem Inter-VM Communication

The isolated ivshmem performance results are depicted in figure 29 and figure 30. The blue curves show the implementation of a fully virtualized (emulated) TCP/IP network, which is still required for initial MPI synchronization. The figures below show inferior performance but far better than the non-SHM solution, depicted in figures 27 and 28. As expected, performance improves if the same tuning measures are engaged, i.e., level-3 caching and virtio instead of OVS. The effect is presented in the gray curves. Additional gains are possible by proper NUMA allocation and CPU pinning.

7.2.2.1 Proper NUMA Allocation - No Load Balancing

Yellow curves of figure 29 and figure 30 demonstrate that manual VM allocation into the same NUMA region results in further performance improvements. In such a case, all VMs have the same access latency to the host's physical shared-memory. On the other hand, allocating VMs to different memory regions leads to frequent cache misses and unpredictable data access times.

7.2.2.2 CPU Load Balancing Active

Suppose the cache-misses occur together with the vCPU rescheduling from one physical processing unit (i.e., core) to another. In that case, the result is a non-monotonously improving performance for increased message size. This behavior is noticeable in the yellow curve of figure 29 for messages ranging from 4B to 1KB. To compensate for measurement fluctuations and to improve CPU load balance, each host vCPU process was manually bound to a specific physical processing unit in a procedure known as CPU pinning. However, the utmost performance boost is achieved if ivshmem is used together with level-3 caching, virtio, proper NUMA allocation, and CPU pinning. These results are demonstrated by the orange curves in figures 29 and 30. In this case, bandwidth reaches 2 GB/s for a message size of 1 MB, which is twice as much as the best TCP/IP case. For smaller message sizes, the difference is even more significant.

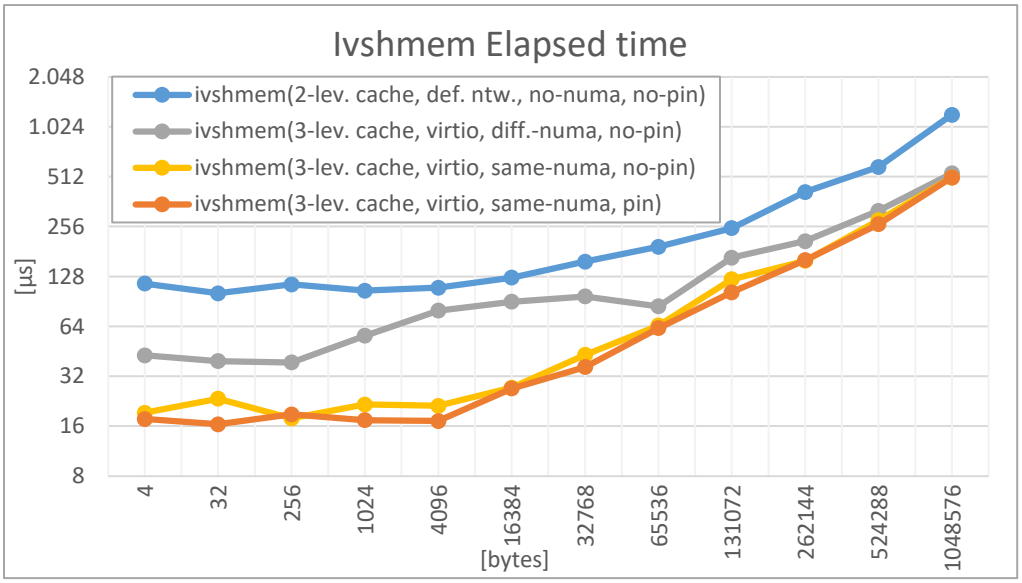

Figure 29: Elapsed times for ivshmem-based inter-VM communication.

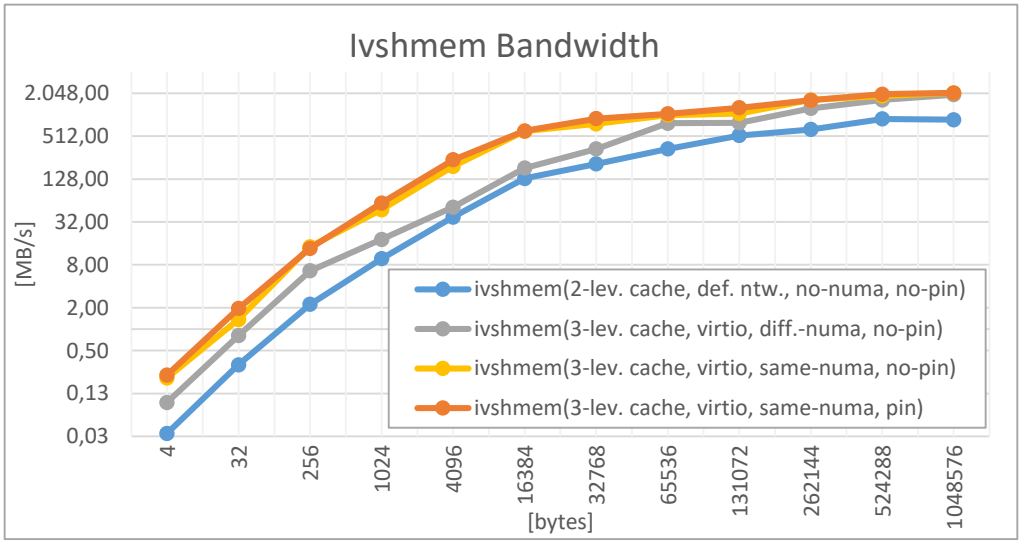

Figure 30: Bandwidth for ivshmem-based inter-VM communication.

The direct comparison of standard TCP/IP OpenStack cloud performance with our ivshmem integration is depicted in figure 31 and figure 32 for elapsed time and bandwidth, respectively. We have also intentionally selected the same NUMA region for inter-VM measurements since Nova randomly performs NUMA allocations, allowing different memory regions during each instance creation.

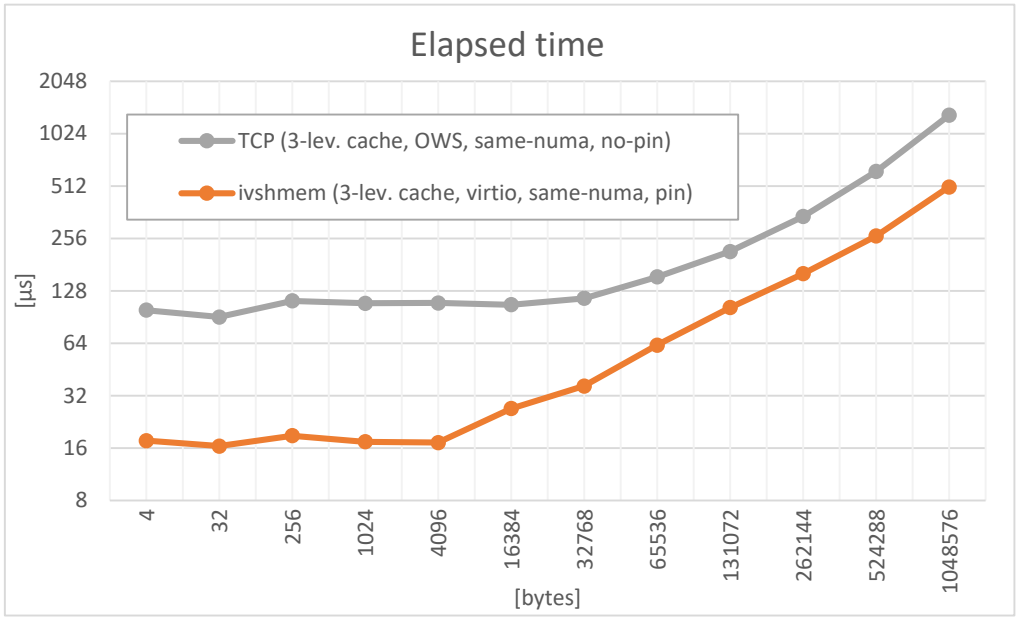

Figure 31: Elapsed times for TCP and ivshmem-based inter-VM communication.

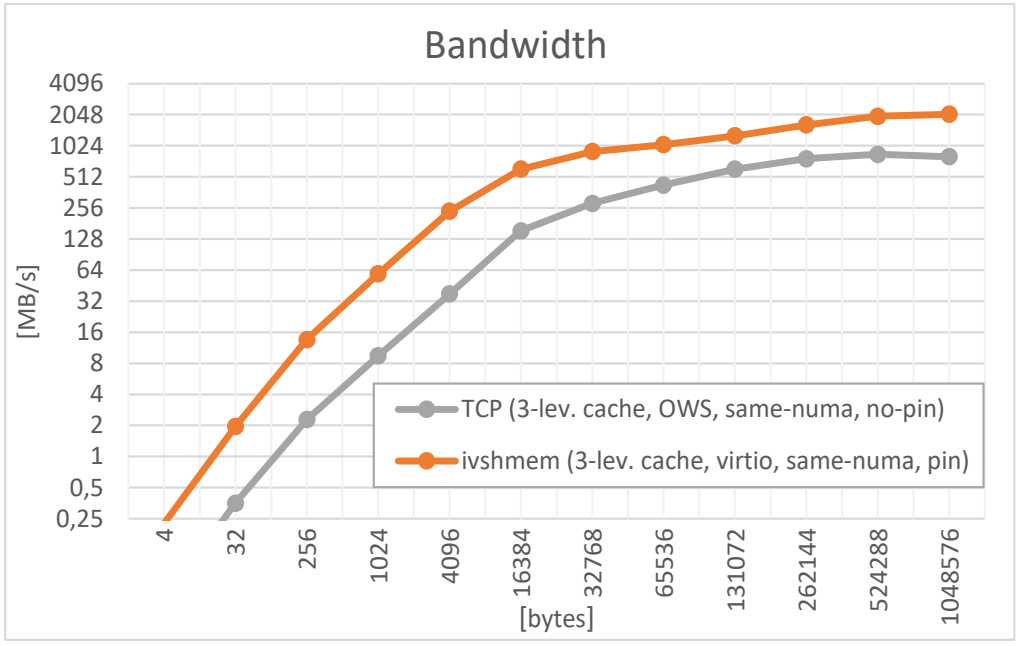

Figure 32: Bandwidth for TCP and ivshmem-based inter-VM communication.

For the smaller size messages, synchronization is dominating communication, and the performance difference is a factor of six in favor of ivshmem. On the other hand, a further increase in message block size leads to data flow becoming the main contributor to overall communication time. From this moment on, the difference between two communication technologies drops to the factor of three for the largest messages. This is a remarkable result, considering that NUMA tuning was not applied, as explained earlier.

7.3 Micromagnetic Simulation in Cloud

As in our initial measurements given in chapter 4, we have used a ferromagnetic material called Permalloy with x/y/z-dimensions of 30x30x100 nm3, respectively. Further input configuration included magnetic saturation of 0.86e6 A/m, damping constant of 0.5 and exchange-coupling constant A=13.0e-12 J/m. As output, the magnetization vector time course was calculated for the interval of 0-300 ps, which is the point in time when the Permalloy reached saturation. Finally, all measurements are performed in our strongest hardware with a focus on inter-VM intra-server communication.

In order to perform the precise estimation of improvements introduced with ivshmem, we have combined three scenarios and present them in Table 5:

1) Restored original Magpar solver with mpich2 communication libraries and MPD process manager.

2) Magpar upgraded to the latest MPI-3 standard with corresponding performance optimizations.

3) Magpar integrated with MVAPICH2-virt and ivshmem.

Furthermore, besides elapsed time T, we have provided in the post-processing phase additional two parameters: speedup S and efficiency E. We calculated them in the following way: S is defined as the execution time on n vCPUs compared to that on 1 vCPU. E is defined as E = S/n and denotes the vCPUs utilization. As seen in Table 5, execution time T is almost identical in all 3 cases when sequential (1vCPU) execution is engaged, which is expected. Since all 3 Magpar runs occurred using internal shared memory enhanced with VT-x hardware accelerator,

visible minor fluctuations could be attributed to the random nature of the host Linux scheduler.

Table 5: Comparison of different Magpar implementations.

	Original (mpich2)			Improved (mpich_3.2)			Ivshmem (mvapich2)		
Setup	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**
2 vCPUs	84,98	1	100	85,07	0,99	99	86,80	0,98	98
4 vCPUs	97,71	0,87	43,5	92,51	0,92	46	62,14	1,37	68,38
6 vCPUs	116,01	0,73	18,25	92,66	0,92	23	52,79	1,61	40,25
8 vCPUs	292,06	0,29	4,83	273,82	0,31	5,17	52,90	1,61	28,83

With parallel execution, we could observe further improvements introduced by the MPI-3 standard, particularly with 4 vCPUs. This is also a setup where the difference between the original mpich2 and new mpich3.2 is clearly visible. However, because of the initial problem segmentation and following extensive IPC, we could not achieve any speedup higher than 1 when TCP/IP is engaged. This fact is important because it demonstrates that unmodified cloud deployments are not suitable for HPC.

Nonetheless, it should be highlighted that our measurement focus was mainly the load of inter-VM IPC, and we have disregarded all parallel intra-VM solver executions. As shown by initial intra-VM measurements (chapter 4), shared memory performance of mpich2-shm and Nemesis-shm channels is sufficient, and further improvements are not expected.

On the other hand, parallel execution based on ivshmem is superior and brings a total speedup of more than 61%. This acceleration is significant, considering that a relatively smaller problem size was used as input. However, with larger numbers of grid points, it is expected that achieved speedup would be superseded several times, as shown in our previous measurements. An additional reason we could not observe more than 100% speedup, e.g., in pure high-performance measurements depicted in chapters 5.1 and 5.2, is that Magpar is a complex micromagnetic solver, and communication is only one segment of overall execution time. In other words, simulation tools such as Magpar spend most of the runtime in the computation phase, especially with larger systems.

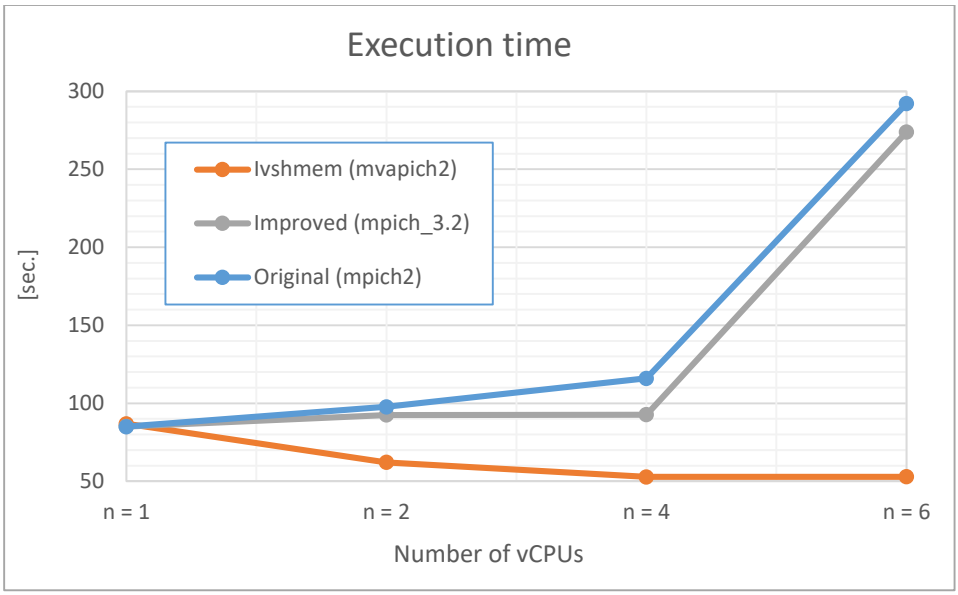

Figure 33: Magpar execution time in relation to the number of vCPUs.

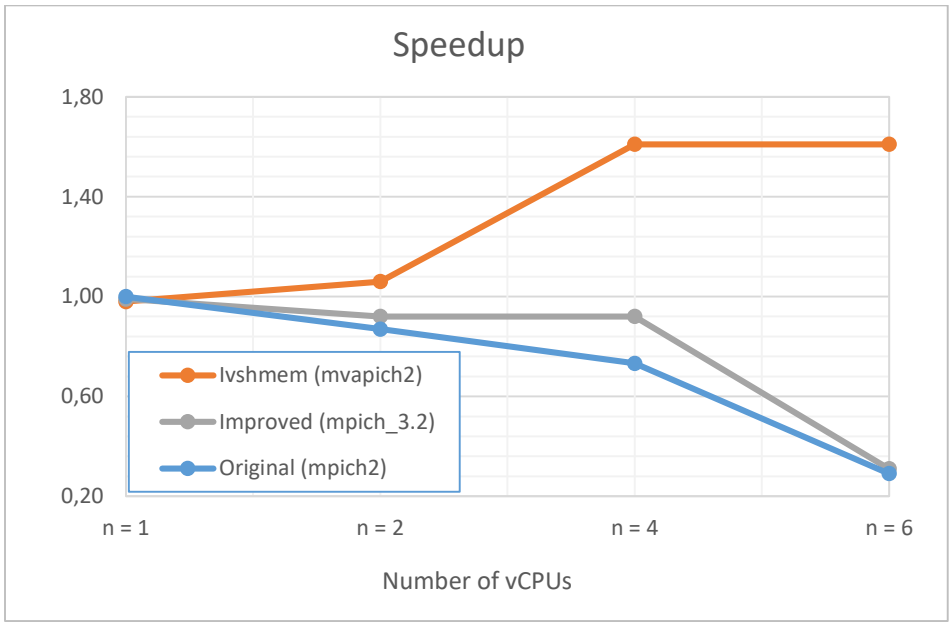

Figure 34: Magpar execution time speedup in relation to the number of vCPUs.

An additional aspect of inter-VM communication shown by these measurements is the saturation of processing elements. In figure 33, one could observe that the best performance is reached already at 4 vCPUs with ivshmem and, to a certain extent, mpich3.2. Therefore, a further increase in parallel processes would only deteriorate achieved performance gains. This notion clearly demonstrates the

complexity of virtualization and the overhead amount created during inter-VM IPC, although internal shared memory utilization benefits were not considered. As expected with complex simulation tools, efficiency E is dropping with each increase of processing elements (vCPUs), while ivshmem remained superior to both TCP/IP competitors.

Finally, ivshmem provided more than six times better efficiency for the maximal number of vCPUs tested. We have also achieved that Magpar was executed for the first time in the OpenStack cloud using ivshmem for inter-VM communication and demonstrated its supremacy over the standard TCP/IP channel. Our tests showed that ivshmem induced performance improvements ranging from 1.4-6, depending on the number of parallel processes, which is remarkable.

7.4 Inter-Server Communication Enhancement

In an effort to maintain consistency with previous measurements, all results and test cases in this chapter are based on the Magpar setup described in the previous section (6.2.3). For a better overview, the results are divided into two chapters. The first one, named MPI-3 and CPU isolation, reflects the improvements made with rebuilding Magpar with the latest MPI-3 standard, which was mpich3.2 at the time of testing. In addition, the CPU load balance optimization technique based on CPU isolation was applied, which solved frequent vCPU de-scheduling issues explained in chapter 6.3.3. Since this procedure requires host access, the following approach to inter-server communication enhancement is more suitable for private cloud infrastructure. Last but not least, all measurement results presented in this section are made with new interconnect hardware based on Infiniband technology.

The second chapter provides an essential comparison of the initial (Ethernet) setup and the original Magpar installation on one side with improved OpenStack hardware-software integration on the other. Also, the broken Magpar was restored to the state closest to the original, which was required for a more precise estimation of introduced changes. Besides these two sets of results, the final (third) one is added as an overall reference point. It emphasizes the combined effect of inter-server communication improvement. The resulting diagrams clearly demonstrate

the importance of cloud modification, mainly because of its inherent weaknesses regarding system scaling.

Finally, it is important to note that adequate MPI process distribution, discussed earlier, is included in all measurements presented in this chapter. Namely, our preliminary tests indicated that the execution efficiency improvement of over 45% could be achieved if a correctly designed host-file is called along with the simulation. This result was obtained for the highest number of processing elements tested (8 vCPUs), whereas its impact grew in accordance with the number of parallel processes. The described comparison was made with the standard mpirun script, excluding the host-file option. This insight is essential for virtual system scaling as contemporary MPI documentation does not indicate these dramatic differences in simulation performance. In addition, an extra attempt to demonstrate the effect of incorrect MPI process distribution was made, yielding the following results: if only one process (e.g., process 2) is scheduled intentionally on the other server, the overall execution time deterioration exceeds 350%, which is entirely unacceptable.

7.4.1 MPI-3 and CPU Isolation

Table 6: Comparison of different Magpar implementations in the inter-server test scenario.

	Original (mpich2)			mpich3.2 (isolcpus_off)			mpich3.2 (isolcpus_on)		
Setup	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**
2 vCPUs	108.82	1	100	106.0	1.03	102.7	101.0	1.08	107.7
4 vCPUs	118.06	0.92	46.1	101.9	1.07	53.5	97.8	1.11	55.5
6 vCPUs	121.92	0.89	29.8	102.6	1.06	35.3	92.9	1.17	39.0
8 vCPUs	145.69	0.75	18.7	112.0	0.97	24.3	89.0	1.22	30.5

Initial values for both original mpich2 as well as the latest mpich3.2 are not that different, although there is a slight advantage in favor of the latter, which is expected. On the other hand, engaging the CPU isolation mechanism described in chapter 6.5.4 introduces an immediate change in execution efficiency that leads to 8% of speedup improvement. This was an important early indication because

point-to-point communication does not induce the cumulative effect of multiple parallel processes.

However, the most significant change occurs if the number of vCPUs is doubled. Similar to our simulation software evaluation tests performed with Ethernet coupled servers, the efficiency of mpich2 dropped sharply under 50% (Table 6), which is very poor. Moreover, a speedup decrease of 8-25% makes this setup inadequate for parallelization as it only deteriorates the execution time. If we follow the vCPU scaling pattern in figure 36 (blue line), we could observe that this trend of speedup deterioration only accelerates with each parallel node addition. As a consequence, the application of Magpar's original parallel libraries for cloud simulations is excluded, at least for LLGE time-step calculation and selected system size.

Compared with obsolete mpich2 based on MPD ring, the enhanced Magpar library, depicted with the gray curve in figure 35, shows clear improvement when 4 vCPU scenario is engaged. Although this change is not astounding in absolute values, it crosses the critical speedup threshold by reaching a 7% improvement over the initial value (Table 6). However, mpich3.2 alone could not bring further execution time acceleration, which is clearly visible in figure 35. This means that other methods should be applied if speedup saturation should be shifted to a much higher number of parallel processing units instead of a very modest 4 vCPUs.

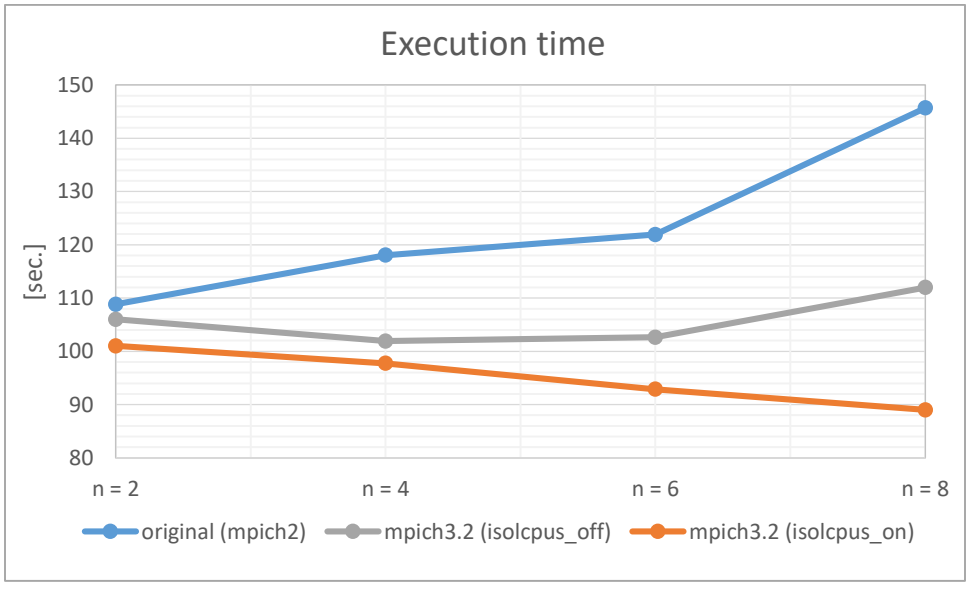

Figure 35: Measurements of elapsed time for various message sizes.

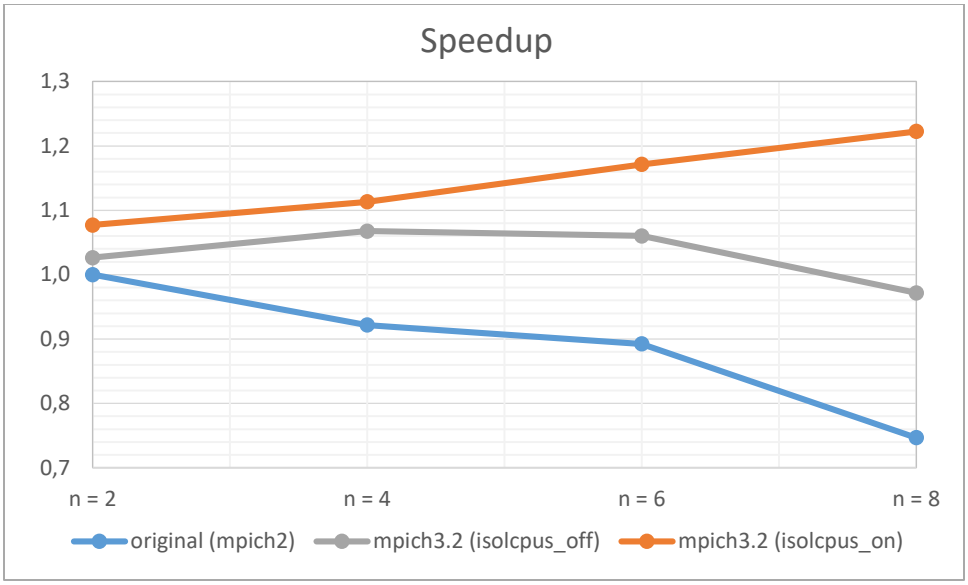

Figure 36: Measurements of speedup for various message sizes.

Besides apparent advantages in both speedup and execution time, engaging the latest iteration of the MPI-3 standard with the CPU isolation mechanism provides a critical advantage required for cloud environment simulation. Namely, as seen in figures 35 and 36 (orange line), this configuration allows continuous scaling of the parallel compute units (vCPUs), which is a necessary precondition for efficient cloud execution. What is also noticeable, the speedup curve (orange) in figure 36 grows almost linearly with the increase of vCPUs, which is a tremendous end-result considering latency between servers.

The measurement results presented in this chapter demonstrate the absolute necessity for continuous parallelization libraries improvements on one side and efficient CPU load balancing on the other. However, only combined, these measures could yield results sufficient for a cloud application.

7.4.2 Ethernet Replacement and Final Comparison

Table 7: Comparison of different Magpar implementations.

	mpich2 (Ethernet)			mpich2 (Infiniband)			mpich3.2 (isolcpus_on)		
Setup	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**	**T [s]**	**S**	**E [%]**
2 vCPUs	176	1	100	108.8	1.62	161.7	101.0	1.74	174.2
4 vCPUs	152	1.16	57.9	118.1	1.49	74.5	97.8	1.80	90.0
6 vCPUs	648	0.27	9.1	121.9	1.44	48.1	92.9	1.89	63.2
8 vCPUs	∞	0	0	145.7	1.21	30.2	89.0	1.98	49.4

The first attempt of Infiniband cloud integration described in the SimPaas project [85] produced poor results in performance upgrades, particularly when estimated through the lenses of bandwidth or latency. Fortunately, OpenStack advancement provided alternative software integration solutions such as VXLAN, allowing a much better perspective for hardware interconnect enhancement and management-network organization. The outlying results, reflecting the change in OpenStack's underlay network, are given in the first two columns of Table 3. The third one, which presents the end-result of our inter-server data exchange improvement, is used as a reference for better evaluation of measurement results. Also, it allows a direct comparison of the initial setup and the final upgrade of Magpar cloud execution.

The first noticeable detail of the Ethernet coupled cloud is a significant difference in execution time for the 2 vCPU, which was expected due to the much lower throughput of the Ethernet Interface. However, the main difference with respect to the execution time assessment occurs when 4 vCPUs (2 per server) are engaged. Namely, the T value drops if the number of parallel processes in Ethernet coupling is doubled (Table 7), which is the trend opposite to one observed in the Infiniband network. Although this seems surprising at first glance, other factors, such as a load of services, software agents, and daemons described in chapter 5.4.4.3, could also influence the CPU load, which is clearly demonstrated with our CPU isolation mechanism (3rd column in Table 7).

The third setup comprising 6 vCPUs, reveals the main characteristic of the original interconnection – utter scaling limitation. The best reflection of this statement could be given with the blue line in figure 37.

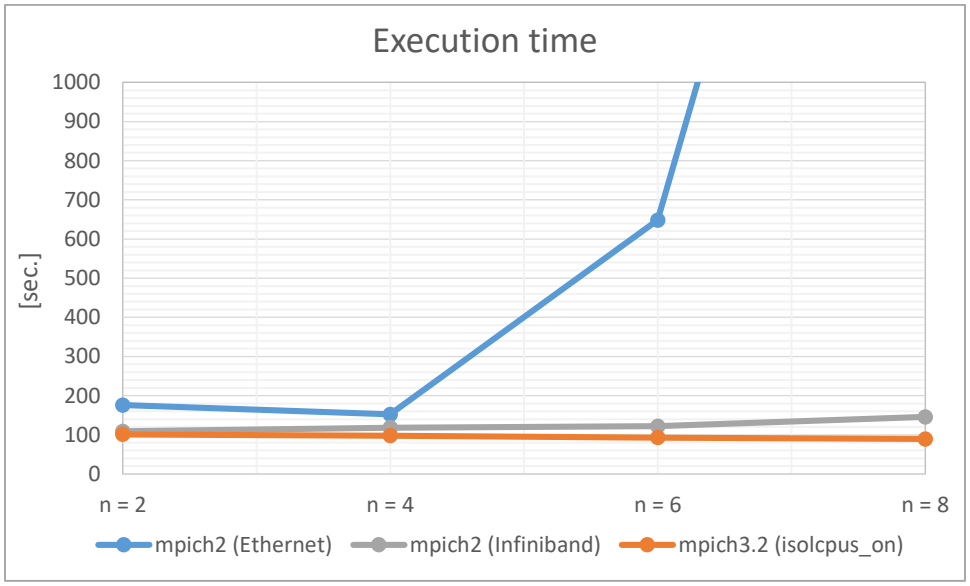

Figure 37: Measurements of elapsed time for various message sizes.

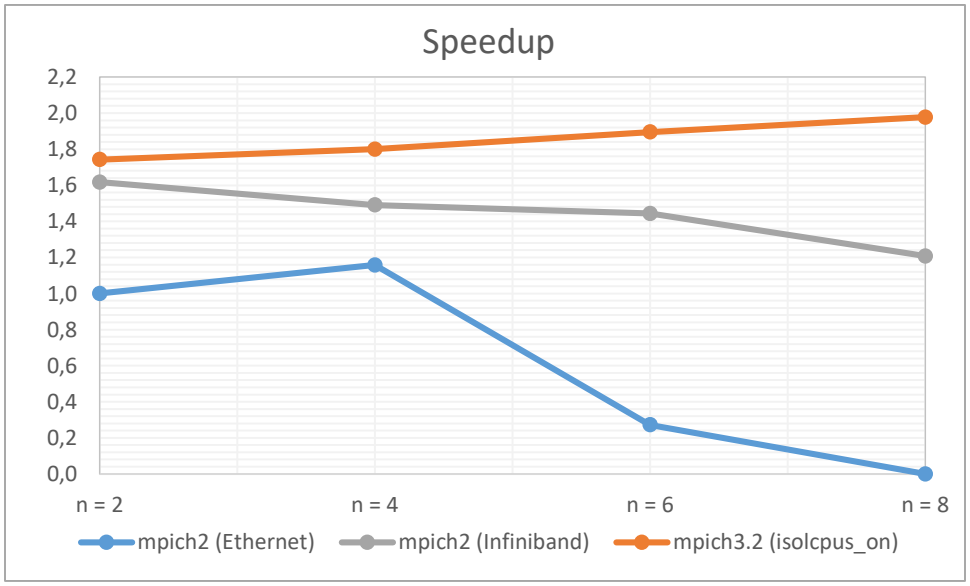

Figure 38: Measurements of speedup for various message sizes.

The Ethernet coupling yielded such bad inter-server communication results that further increase of parallel processes led even to the solver blocking for an unknown time. This obstruction is the reason why execution time is shown as infinite in Table 7, for the case of 8 vCPUs. On the other hand, Infiniband implementation is far better in every respect (figure 38), but these results alone are insufficient for a cloud simulation. As seen in the second column of Table 7, execution time T increases with each increment of parallel processes, making this setup useless for cloud execution.

Finally, our solution that combines new hardware interconnection, upgraded MPI parallel libraries, proper MPI process distribution, and efficient CPU load balancing allows for the first time the necessary speedup for inter-server cloud simulation of micromagnetic materials. Besides achieving almost two times the initial execution acceleration, our solution brings almost linear speedup development, which is a significant result considering the relatively small system size selected for LLGE calculation.

7.5 Code Parallelization and Scaling Limitations

In order to evaluate the scaling capabilities of simulation software in a complex environment such as cloud OS, other internal code parameters have to be included in the assessment. The most important parameter related to the fixed problem size scaling or the so-called weak scaling is the ratio between parallelized and non-parallelized parts of the code. Namely, the percent of sequential code within the solver is a limiting factor to the maximal theoretical speedup. Since this limitation is determined in ideal conditions, i.e., scaling efficiency remains 100%, the actual measured values will be substantially smaller.

To better illustrate the described problem, we will use a code with a serial/parallel ratio of 1:1 or 50% each, presented in figure 37. Because only the parallel code part could be scaled (case of 2 processors), the total execution time could be reduced by only 25% or one-half of parallel code execution time.

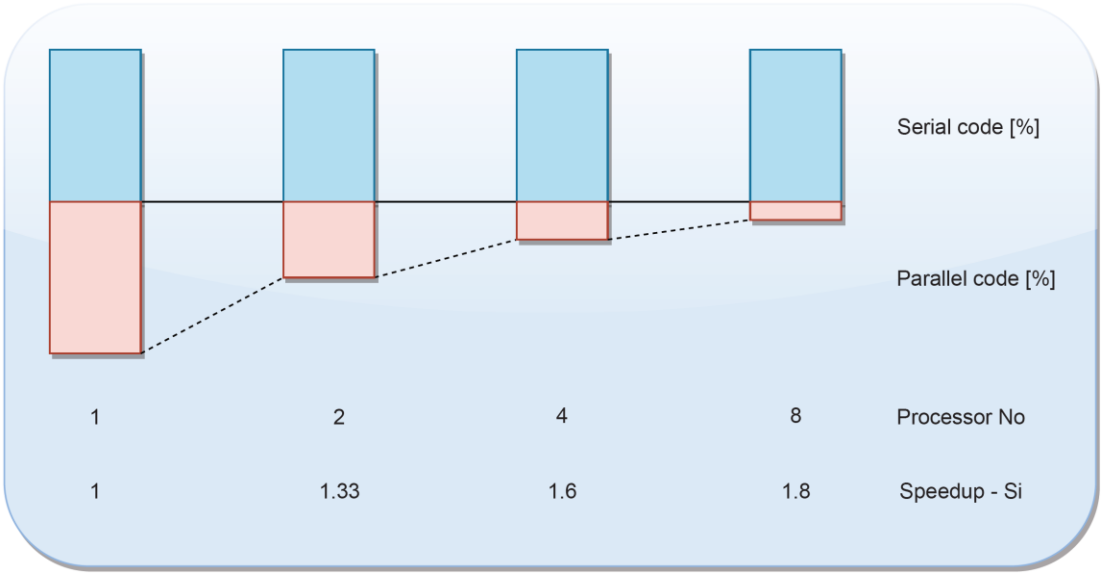

Figure 37: Parallel/Serial code scaling for a fixed size problem.

The same principle applies if we continue to scale up the number of processors, which could be seen in the graphical representation of figure 37. Of course, these calculations are applicable if scaling efficiency is always 100%, which is not the case in real measurements. From this example, it is clear that the serial code presents a decisive limitation factor, even when the parallel code execution time becomes infinitely small. This problem was recognized by American computer expert Gene Amdahl in 1967 [90], and it is known today as Amdahl's law.

The definition of Amdahl's law is as follows:

Eq. (3): $$Si = \frac{T1}{Ti} = \frac{p+n}{n+{}^{p}/_{i}} = \frac{1}{1-p+{}^{p}/_{i}}$$

Si is speedup for the i-number of processors, p is the percent of parallel execution time, and n – percent of non-parallel (sequential) time. Please also note that since Si is defined as a ratio, we could easily switch from absolute time values (T1, Ti) to a relative, i.e., percentages (p, n). However, the main conclusion derived from Amdahl's law is drawn in case of an extremely high number of processing elements (equation 4):

Eq. (4): $$\lim_{i \to \infty} S(i) = \frac{1}{1-p} = \frac{1}{n}$$

From equation 4, it is clear that the non-parallel portion (n) of any simulation software is a vital scaling limitation factor, particularly if the system size is not changing. Alternatively, one could increase the problem size and avoid these limitations, but it requires a change of measurement purpose, which was not our goal.

In the example given above (figure 37), the maximal theoretical speedup is 2, or an inverse value of 50% (0.5), which is very poor. Moreover, even with a significantly higher parallel proportion p, for instance, 95%, the maximum speedup (Si) is 20, which refers to ideal conditions that could not be met in practice. The other factors, such as processor speed, cache size, CPU load balancing, and perhaps the most significant communication overhead, are the ones that determine the real cloud scaling parameters. In the next chapter, we will estimate the possible values of p in our micromagnetic simulation of choice, Magpar, and evaluate its influence on cloud simulation performance.

7.5.1 Magpar and Micromagnetic Solver Parallelization

In an effort to perform precise solver performance estimation during parallel execution, knowing proportions of p or n (equation 3) is vital to accurate analyses. Ideally, this information should be provided by the code developers, but in practice, it is often not the case. Alternatively, one could develop an approximate method for a range of p (or n) values of interest and manually estimate the impact on attained measurement results. However, specific requirements have to be met or assumed:

1) p has to be calculated from the measured values of Si
2) The maximal possible number of processors have to be used
3) CPU time has to be measured instead of wall time
4) The problem size and simulation input have to remain the same during tests
5) CPU load balancing has to be applied for consistent results

6) CPUs across the clusters need to have the same cache and memory access characteristics
7) Solver initiation time has to be negligible, or alternatively, the initiation/solving ratio has to be known

From the list above, it is clear that meeting all these requirements will be challenging, and in many cases, impossible, especially if the hardware required for these kinds of tests is not available. On the other hand, this precision level is mandatory for extreme scaling, which is not the case in the cloud. Fortunately, the author of Magpar published benchmark results [91] that are sufficient for our analyses.

Therefore, the first and obvious step would be calculating the value of p by using measured values of speedup (Si) for a different number of processors (i). For this purpose, we will use equation 5, which is derived directly from equation 3.

Eq. (5): $$p = \frac{1-\frac{1}{Si}}{1-\frac{1}{i}}$$

The results obtained from [91] and equation 5 are setting the range of p between 90–98.5%, which is more than enough for our Magpar cloud performance estimation. As a result, even with the lowest values of calculated p, the theoretically achievable speedup is far greater than the values we measured in a cloud. All things considered, it is safe to conclude that the other factors, such as the 3-level communication hierarchy, for instance, are playing a crucial role in determining the upper-performance limitations of any cloud simulation.

8 CONCLUSION

This thesis considers magnetic nanomaterials, simulation system modeling, and corresponding solvers for computation and efficient cloud execution. By this manifold approach, substantial insights are achieved to improve existing solvers on the communication side, remove obstacles for efficient cloud execution, and explore effective measures for turning the virtual environment into an HPC platform. However, simulation performance strongly depends on the cloud's inter-server, inter-VM, and inter-vCPU bandwidth and latency. If these metrics are insufficient, a standard cloud will not speed-up but slow-down parallel codes because communication overhead between VM threads cannot always be minimized. The reason for that behavior lies in the plethora of software interfaces that exist in a standard cloud between interacting parts of a parallel code.

However, we can substantially reduce the OpenStack's overhead during inter-VM, inter-server, or inter-vCPU communication. We achieve this without modifying the Linux kernel, QEMU, KVM, or cloud OS. Furthermore, OpenStack segments could be excluded from HPC communication in some cases, which is a significant advantage related to cloud efficiency. For that purpose, the inter-VM shared-memory device - ivshmem is engaged, allowing guest MPI libraries direct access to the host shared memory. This approach suggests the ivshmem creation between VMs of the same server, which dramatically improves the data exchange rates. Last but not least, we integrated vendor-specific Infiniband drivers with guest simulation software by adapting it to different versions of MPI libraries, albeit keeping an Ethernet connection for management and internet access. The project enrolled in three stages, reflecting predefined milestones required for successful micromagnetic cloud integration.

In the first stage of our cloud optimization for HPC, we have rebuilt the spinlock-free access synchronization of ivshmem, which is necessary for efficient inter-VM intra-server communication. Since ivshmem is based on shared memory and not TCP/IP, it could deliver much better performance. We have also managed to integrate ivshmem with the latest QEMU, libvirt, MPICH, and Linux versions and provided comprehensive documentation. Moreover, we have created an HPC

performance measurement tool by integrating custom user-level UIO drivers with the MPICH programming interface. As a result, our adaptation of the MPICH communication mechanism transparently replaced a one-sided MPI_PUT call with an ivshmem-based wrapper function. We have also shown that ivshmem is superior throughout the whole range of message sizes to all TCP/IP-based communication channels. Accordingly, significant performance gains are demonstrated, ranging from 3 to a factor of 10, depending on the message size. For messages greater than 512kB and particular hardware configuration, ivshmem even surpasses non-virtualized host SHM performance, which is a remarkable result.

With completely restored and tested ivshmem shared memory communication mechanisms, we have proceeded to the second stage of our private cloud optimization – OpenStack integration. As one of our main results, we have seamlessly implemented ivshmem into the contemporary version of OpenStack and then further improved the existing virtual network with custom virtio Bridge. Additionally, we have introduced several tuning methods, such as NUMA aware scheduling, CPU load balancing, MPI process distribution planning, and therefore, adapted ivshmem to perform even better. Finally, we have conducted practical measurements in carefully selected configuration setups for both ivshmem and TCP/IP-based emulation of physical shared-memory. All setups and methods are then evaluated and compared, showing that the best ivshmem scenario is at least three times as fast as the best TCP/IP case. As in stage one, we have used for measurements our wrapper versions of MPICH's MPI_PUT for data-exchange and MPI_WIN_LOCK for shared-memory synchronization.

In the final stage, after the successful integration of our private OpenStack cloud with ivshmem, we have focused on the primary goal of our project – enabling efficient cloud simulation of magnetic nanomaterials. However, our best micromagnetic candidate Magpar was broken and had to be restored first. Furthermore, we have improved the original version by upgrading the communication libraries to the most recent MPI-3 standard. This change introduced a more optimized Nemesis channel and replaced the obsolete MPD process manager with the latest, user-friendly successor called Hydra. Finally, in the last step, we have merged Magpar with ivshmem using the MVAPICH2-virt library and demonstrated its superiority over the standard TCP/IP channel. As a

result, we have achieved that Magpar runs for the first time in OpenStack using ivshmem for inter-VM communication, with performance improvements between factors of 1.4-6, depending on the degree of parallelization. Also, inter-server communication received significant upgrades with respect to the execution acceleration, allowing for the first time near-linear increase of simulation speedup with each doubling of the parallel nodes. The overall speedup reached almost two times better value than the original setup, which is a remarkable result considering the substantial latency difference between cables and host memory.

However, the most significant benefit of using a cloud for running micromagnetic simulations is rapid system scaling and efficient provisioning of the incorporated software setups. The latter is very important because one of the main problems in collaboration between international or even regional researchers is the inability to share the simulation execution environment. With the cloud approach described in this thesis, this colossal obstacle is easily removed. By engaging a simple VM snapshot, scientists could quickly exchange created disc images with all preinstalled software packages, files, libraries, and data structures. Finally, cloud-based simulation is not a direct competitor to existing supercomputers in terms of sheer power but rather a cost-efficient alternative with additional features that could decide the perspective computing platform selection.

BIBLIOGRAPHY

[1] W. Scholz, J. Fidler, T. Schrefl, D. Suess, R. Dittrich, H. Forster and V. Tsiantos, "Scalable Parallel Micromagnetic Solvers for Magnetic Nanostructures," in *Comp. Mat. Sci.*, 2003.

[2] H. Fangohr, T. Fischbacher and M. Franchin, "University of Southampton," NMAG User Manual (0.2.1), 2014. [Online]. Available: http://nmag.soton.ac.uk/nmag/current/manual/singlehtml/manual.html. [Accessed September 2020].

[3] National Institute of Standards and Technology, "The Object Oriented MicroMagnetic Framework (OOMMF) project at ITL/NIST," [Online]. Available: https://math.nist.gov/oommf/. [Accessed September 2020].

[4] University of York, "Vampire - Atomistic simulation of magnetic materials," [Online]. Available: http://vampire.york.ac.uk/. [Accessed September 2020].

[5] D. Revele, "Hypervisors and Virtual Machines Implementation Insights on the x86 Architecture," October 2011. [Online]. Available: https://www.usenix.org/system/files/login/articles/105498-Revelle.pdf. [Accessed September 2020].

[6] F. Bellard, "QEMU System Emulation User's Guide," 2020. [Online]. Available: https://www.qemu.org/docs/master/system/index.html. [Accessed September 2020].

[7] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Scheduling_(computing). [Accessed September 2020].

[8] J. Denton, Learning OpenStack Networking: Build a solid foundation in virtual networking technologies for OpenStack-based clouds, 3rd Edition, Birmingham: Packt Publishing Ltd., 2018.

[9] "OpenStack," OpenStack Foundation, [Online]. Available: https://www.openstack.org/. [Accessed September 2020].

[10] W. Gropp, E. Lusk and A. Skjellum, Using MPI Portable Parallel Programming with the Message-Passing Interface, Third Edition, Cambridge: The MIT Press, 2014.

[11] A. C. Macdonell, "Shared-Memory Optimizations for Virtual Machines," University of Alberta, Edmonton, Alberta, 2011.

[12] P. Ivanovic and H. Richter, "Performance Analysis of ivshmem for High-Performance Computing in Virtual Machines," *Journal of Physics: Conference Series,* vol. 960, p. 012016, 2018.

[13] P. Ivanovic, H. Richter and A. Bozorgmehr, "Cloud-Efficient Modelling and Simulation of Magnetic Nano Materials," Simulationswissenschaftliches Zentrum Clausthal-Göttingen, Clausthal/Göttingen, 2017.

[14] H.-J. Koch, "The Userspace I/O HOWTO," 11 December 2006. [Online]. Available: https://www.kernel.org/doc/html/v4.14/driver-api/uio-howto.html. [Accessed September 2020].

[15] N. Manchanda and K. Anand, " Non-Uniform Memory Access (NUMA)," in *New York University*, 2010.

[16] P. Ivanovic and H. Richter, "Shared Memory Enhanced Cloud as a Computing Tool for Micromagnetic Simulations," in *ACM: The 5th International Conference on Computing and Data Engineering (ICCDE 2019)*, Shanghai, China, 2019.

[17] P. Ivanovic and C. Siemers, "Private Cloud Tuning for Efficient Inter-Server Simulation Execution," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, USA, 2021.

[18] P. Ivanovic and H. Richter, "OpenStack Cloud Tuning for High Performance Computing," in *Proc. 3rd IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA 2018)*, Chengdu, China, 2018.

[19] Wikipedia, "Nanomaterials," [Online]. Available: https://en.wikipedia.org/wiki/Nanomaterials. [Accessed September 2020].

[20] J. Akerman, "Toward a Universal Memory," in *American Association for the Advancement of Science*, 2005.

[21] R. D. C. and S. M. D., "Erratum to “Spin Transfer Torques” [J. Magn. Magn. Mater. 320 (2008) 1190-1216]," in *Journal of Magnetism and Magnetic Materials, Volume 321, Issue 16*, 2009.

[22] K. Jorissen, F. Vila and J. Rehr, "A high performance scientific cloud computing environment for materials simulations," *Computer Physics Communications,* vol. 183, no. 9, pp. 1911-1919, 2012.

[23] S. C. -. Göttingen, "A Cloud-based Software Infrastructure for Distributed Simulation," [Online]. Available: https://www.simzentrum.de/en/education/cloud-basierte-software-infrastruktur-fuer-verteilte-simulation/. [Accessed December 2020].

[24] C. Jermain, G. Rowlands, G. Buhrman and D. C. Ralph, "GPU-accelerated micromagnetic simulations using cloud computing," *Journal of Magnetism and Magnetic Materials,* vol. 401, pp. 320-322, 2016.

[25] A. Vansteenkiste, "mumax3," Ghent University, Belgium, [Online]. Available: https://mumax.github.io/. [Accessed December 2020].

[26] GoParallel S.L., "GPMagnet Parallel Micromagnetics Problems Solver," [Online]. Available: http://www.goparallel.net/documents/GPMagnet%20-%20Computer%20Requirements.pdf. [Accessed December 2020].

[27] Mellanox Technologies Inc, "Introduction to InfiniBand," [Online]. Available: https://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf. [Accessed September 2020].

[28] Wikipedia, "Landau–Lifshitz–Gilbert equation," [Online]. Available: https://en.wikipedia.org/wiki/Landau%E2%80%93Lifshitz%E2%80%93Gilbert_equation. [Accessed September 2020].

[29] University of Southampton (UK) and European XFEL GmbH (Germany)., "FinMag: finite-element micromagnetic simulation tool," [Online]. Available: https://github.com/fangohr/finmag. [Accessed December 2020].

[30] D. Berkov, "MicroMagus Software package for micromagnetic simulations," [Online]. Available: http://www.micromagus.de/home.html. [Accessed December 2020].

[31] D. Suess and T. Schrefl, "SuessCo Simulations," SuessCo KG, [Online]. Available: https://www.suessco.com/simulations/. [Accessed December 2020].

[32] Wikipedia, "Paramagnetism," [Online]. Available: https://en.wikipedia.org/wiki/Paramagnetism. [Accessed September 2020].

[33] Wikipedia, "Diamagnetism," [Online]. Available: https://en.wikipedia.org/wiki/Diamagnetism. [Accessed September 2020].

[34] Wikipedia, "Antiferromagnetism," [Online]. Available: https://en.wikipedia.org/wiki/Antiferromagnetism. [Accessed September 2020].

[35] P. Weiss, "Magnetic domain," [Online]. Available: https://en.wikipedia.org/wiki/Magnetic_domain. [Accessed September 2020].

[36] University of Minnesota, "Lower Switching Current for Spin-Torque Transfer in Magnetic Storage Devices such as Magnetoresistive Random Access Memory (MRAM)," 2020. [Online]. Available: http://license.umn.edu/technologies/z09007_lower-switching-current-for-spin-torque-transfer-in-magnetic-storage-devices-such-as-magnetoresistive-random-access-memory-mram. [Accessed September 2020].

[37] A. Drews, G. Selke, B. Krueger, C. Abert and T. Gerhardt, "MAGNUM.fd," [Online] Available: http://micromagnetics.org/magnum.fd/index.html. [Accessed December 2020].

[38] T. Mattson, H. He and A. Koniges, The OpenMP Common Core Making OpenMP Simple Again, The MIT Press, 2019.

[39] Software in the Public Interest, "Open MPI: Open Source High Performance Computing," [Online]. Available: https://www.open-mpi.org/. [Accessed September 2020].

[40] A. Amer, P. Balaji, W. Bland, W. Gropp, R. Latham and H. Lu, "MPICH User's Guide, Version 3.2.1," 10 November 2017. [Online]. Available: https://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1-userguide.pdf. [Accessed September 2020].

[41] The Ohio State University, "MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE," 2020. [Online]. Available: https://mvapich.cse.ohio-state.edu/. [Accessed September 2020].

[42] P. Balaji, D. Buntinas, R. Butler, A. Chan, D. Goodell, W. Gropp, J. Krishna and R. Latham, "MPICH2 Installer's Guide, Version 1.3.1," 17 November 2010. [Online]. Available: https://www.mpich.org/static/downloads/1.3.1/mpich2-1.3.1-installguide.pdf. [Accessed September 2020].

[43] C. Geuzaine and J.-F. Remacle, "Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities," in *International Journal for Numerical Methods in Engineering 79(11)*, 2009.

[44] "Netgen Mesh Generator," 2020. [Online]. Available: https://sourceforge.net/p/netgen-mesher/wiki/Home/. [Accessed September 2020].

[45] The Apache Software Foundation, "Apache CloudStack," [Online]. Available: https://cloudstack.apache.org/. [Accessed December 2020].

[46] Eucalyptus Systems, Inc., "Eucalyptus," [Online]. Available: https://www.eucalyptus.cloud/. [Accessed December 2020].

[47] OpenNebula Systems, "OpenNebula New True Hybrid Cloud Architecture," [Online]. Available: https://opennebula.io/. [Accessed December 2020].

[48] OpenStack Foundation, "OpenStack Compute (nova)," March 2020. [Online]. Available: https://docs.openstack.org/nova/latest/. [Accessed September 2020].

[49] OpenStack Foundation, "Horizon: The OpenStack Dashboard Project," June 2019. [Online]. Available: https://docs.openstack.org/horizon/latest/. [Accessed September 2020].

[50] O. Foundation, "Welcome to Neutron's documentation!," January 2020. [Online]. Available: https://docs.openstack.org/neutron/latest/. [Accessed September 2020].

[51] OpenStack Foundation, "Welcome to Glance's documentation!," August 2019. [Online]. Available: https://docs.openstack.org/glance/latest/. [Accessed September 2020].

[52] OpenStack Foundation, "Keystone, the OpenStack Identity Service," July 2019. [Online]. Available: https://docs.openstack.org/keystone/latest/. [Accessed September 2020].

[53] M. Jones , "Anatomy of the libvirt virtualization library," 5 January 2010. [Online]. Available: https://developer.ibm.com/technologies/linux/tutorials/l-libvirt/. [Accessed September 2020].

[54] Canonical Ltd., "Infrastructure for container projects," [Online]. Available: https://linuxcontainers.org/. [Accessed September 2020].

[55] B. Kamal, A. El Fergougui and A. Elbelrhiti Elalaoui, "Software-defined networking (SDN): A survey," 2017.

[56] OpenStack Foundation, "Scenario: Classic with Open vSwitch," November 2016. [Online]. Available: https://docs.openstack.org/liberty/networking-guide/scenario-classic-ovs.html. [Accessed September 2020].

[57] OpenStack Foundation, "OpenStack Block Storage (Cinder) documentation," March 2020. [Online]. Available: https://docs.openstack.org/cinder/latest/. [Accessed September 2020].

[58] OpenID Foundation, "What is OpenID?," 2020. [Online]. Available: https://openid.net/what-is-openid/. [Accessed September 2020].

[59] D. Hardt, "The OAuth 2.0 Authorization Framework," October 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749. [Accessed September 2020].

[60] VMware, "Server Consolidation," 2020. [Online]. Available: https://www.vmware.com/de/solutions/consolidation.html. [Accessed September 2020].

[61] P. Kutch and B. Johnson, "SR-IOV for NFV SolutionsPractical Considerations and Thoughts," February 2017. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/sr-iov-nfv-tech-brief.pdf. [Accessed September 2020].

[62] S. Mittal, "A survey of techniques for architecting TLBs," 2016.

[63] Wikipedia, "Memory management unit," May 2020. [Online]. Available: https://en.wikipedia.org/wiki/Memory_management_unit. [Accessed September 2020].

[64] OASIS Open, "Virtual I/O Device (VIRTIO) Version 1.0," December 2013. [Online]. Available: http://docs.oasis-open.org/virtio/virtio/v1.0/csprd01/virtio-v1.0-csprd01.pdf. [Accessed September 2020].

[65] "Xen Project Software Overview," November 2018. [Online]. Available: https://wiki.xen.org/wiki/Xen_Project_Software_Overview#Introduction_to_Xen_Architecture. [Accessed September 2020].

[66] R. Russel, "virtio: towards a de-facto standard for virtual I/O devices," *ACM SIGOPS Operating Systems Review,* vol. 42, no. 5, 2008.

[67] Virginia Tech, "Popcorn Linux," [Online]. Available: http://www.popcornlinux.org/. [Accessed December 2020].

[68] Q. Zhang and L. Liu, "Shared Memory Optimization in Virtualized Cloud," in *2015 IEEE 8th International Conference on Cloud Computing*, New York, NY, USA, 2015.

[69] The Ohio State University, "MVAPICH2-Virt 2.2," Network-Based Computing Laboratory, 2020. [Online]. Available: http://mvapich.cse.ohio-state.edu/userguide/virt/. [Accessed September 2020].

[70] F. Hantelmann, LINUX Start-up Guide, Springer-Verlag Berlin Heidelberg, 1997.

[71] S. Henry-Stocker, "Linux dominates supercomputing," July 2020. [Online]. Available: https://www.networkworld.com/article/3568616/linux-dominates-supercomputing.html. [Accessed September 2020].

[72] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller and M. Upton, "Hyper-Threading Technology Architecture and Microarchitecture," in *Intel Technology Journal Q1*, 2002.

[73] Wikipedia, "Security-Enhanced Linux," April 2020. [Online]. Available: https://en.wikipedia.org/wiki/Security-Enhanced_Linux. [Accessed September 2020].

[74] Canonical, "AppArmor: Linux kernel security module," [Online]. Available: https://apparmor.net/. [Accessed September 2020].

[75] D. C. Latham , "Department of Defense Trusted Computer System Evaluation Criteria," December 1985. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/dod85.pdf. [Accessed September 2020].

[76] P. J. Salzman, M. Burian and O. Pomerantz, "Linux Kernel Module Programming Guide," 18 May 2007. [Online]. Available: ftp://ftp.wayne.edu/ldp/en/lkmpg-2.6/lkmpg-2.6.pdf. [Accessed September 2020].

[77] J. Jose, O. Sujisha, M. Gilesh and T. Bindima, "On the Fairness of Linux O(1) Scheduler," in *5th International Conference on Intelligent Systems, Modelling and Simulation*, Langkawi, 2014.

[78] M. Kerrisk, "eventfd(2) — Linux manual page," June 2020. [Online]. Available: https://www.man7.org/linux/man-pages/man2/eventfd.2.html. [Accessed September 2020].

[79] OpenFabrics Alliance, "OFA Overview," 2018. [Online]. Available: https://www.openfabrics.org/ofa-overview/. [Accessed September 2020].

[80] Intel Corporation, "Intel® Virtualization Technology for Directed I/O (VT-d): Enhancing Intel platforms for efficient virtualization of I/O devices," 03 June 2012. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-virtualization-

technology-for-directed-io-vt-d-enhancing-intel-platforms-for-efficient-virtualization-of-io-devices.html. [Accessed September 2020].

[81] Wikipedia, "Parallels Workstation," February 2020. [Online]. Available: https://en.wikipedia.org/wiki/Parallels_Workstation. [Accessed September 2020].

[82] E. A. Clark, "Ferromagnetic Materials, vol 1," Wolfhart, E.P ed., Amsterdam, p. 531.

[83] C. S. Pabla, "Completely Fair Scheduler," Linux Jurnal, 1 August 2009. [Online]. Available: https://www.linuxjournal.com/node/10267. [Accessed September 2020].

[84] S. Pakin, "Myrinet," in *Encyclopedia of Parallel Computing*, Boston, Springer, 2011.

[85] H. Richter, A. Keidel and R. Ledyayev, "Über die Eignung von Clouds für das Hochleistungsrechnen (HPC)," in *IfI Technical Report Series ISSN 1860-8477*, Germany, 2015.

[86] D. Farinacci, T. Li, S. Hanks, D. Meyer and P. Traina, "Generic Routing Encapsulation (GRE)," March 2000. [Online]. Available: https://tools.ietf.org/html/rfc2784. [Accessed September 2020].

[87] . J. Chu and V. Kashyap, "Transmission of IP over InfiniBand (IPoIB)," April 2006. [Online]. Available: https://www.ietf.org/rfc/rfc4391.txt. [Accessed September 2020].

[88] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," August 2014. [Online]. Available: https://datatracker.ietf.org/doc/rfc7348/?include_text=1. [Accessed September 2020].

[89] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault and R. Namyst, "hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications," in *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010)*, Pisa, Italia, 2010.

[90] A. Gene, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," in *AFIPS Conference Proceedings (30)*, 1967.

[91] W. Scholz , "Magpar Performance," 2010. [Online]. Available: http://www.magpar.net/static/magpar/doc/html/performance.html. [Accessed September 2020].

ABBREVIATIONS

API – Application Programming Interface

BAR – Base Address Register

CFS – Completely Fair Scheduler

CLI – Command Line Interface

CPU – Central Processing Unit

EPT – Extended Page Tables

HPC – High Performance Computing

HW – Hardware

I/O – Input Output

IP – Internet Protocol

IPC – Inter-Process Communication

ISA – Instruction Set Architecture

IVSHMEM – Inter VM Shared Memory

KVM – Kernel-based Virtual Machine

LAN – Local Area Network

LLGE – Landau Lifshitz Gilbert Equation

MPD – Multi-Purpose Daemon

MPI – Message Passing Interface

NIC – Network Interface Card

NUMA – Non-Uniform Memory Access

OFED – Open Fabrics Enterprise Distribution

OS – Operating System

OVS – Open vSwitch

PCI – Peripheral Component Interconnect

PID – Process ID

QEMU – Quick Emulator

RAM – Random Access Memory

RDMA – Remote Direct Memory Access

SMS – Shared Memory Server

SR-IOV – Single Root Input Output Virtualization

SW – Software

TCP – Transmission Control Protocol

UIO – User Input Output

VIRTIO – Virtual Input Output

VM – Virtual Machine

VMCS – Virtual Machine Control Structure

VMX – Virtual Machine eXtension

VXLAN – Virtual Extensible LAN

LIST OF FIGURES

LIST OF TABLES

PUBLICATIONS

- P. Ivanovic, H. Richter, A. Bozorgmehr, Cloud-Efficient Modelling and Simulation of Magnetic Nano Materials, Bericht 2015-2016, Simulationswissenschaftliches Zentrum Clausthal-Göttingen, editors: A. Herzog, T. Hanschke, www.simzentrum.de, 2017.

- P. Ivanovic, H. Richter, High-Performance Computing and Simulation in Clouds, Clausthal-Göttingen International Workshop on Simulation Science (SimScience 2017), Göttingen, Germany, 2017.

- P. Ivanovic and H. Richter, "Performance Analysis of ivshmem for High-Performance Computing in Virtual Machines," *Journal of Physics: Conference Series,* vol. 960, p. 012016, 2018.

  P. Ivanovic and H. Richter, “OpenStack Cloud Tuning for High Performance Computing,” in Proc. 3rd IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA 2018), pp. 142-146, Chengdu, China, 2018.

- P. Ivanovic, H. Richter, Cloud-Efficient Modelling and Simulation of Magnetic Nano Materials, Bericht 2017-2018, Simulationswissenschaftliches Zentrum Clausthal-Göttingen, editors: A. Herzog, T. Hanschke, www.simzentrum.de, 2019.

- P. Ivanovic, H. Richter, Shared Memory Enhanced Cloud as a Computing Tool for Micromagnetic Simulations, ACM: The 5th International Conference on Computing and Data Engineering (ICCDE 2019), pp. 95–99, Shanghai, China, 2019.

- P. Ivanovic and C. Siemers, "Private Cloud Tuning for Efficient Inter-Server Simulation Execution," in 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, USA, 2021.